Universidad Carlos III de Madrid

Escuela Politécnica Superior

Grado en Ingeniería Telemática

Desarrollo de una aplicación web para el proceso de contratación

Trabajo Fin de Grado

Autor: Alberto Martín Casado
Tutor: Pablo Serrano Yáñez-Mingot

Septiembre de 2014

Agradecimientos

Agradecer a mis padres en primer lugar por haberme brindado la posibilidad de cursar este grado y por su infinito apoyo. A mi madre por haberme inculcado los valores morales que me han hecho ser quien soy y por enseñarme el valor de las cosas.

A Rebeca, de la que tanto he aprendido durante los dos años que he trabajado con ella y por todos sus sabios consejos. A Christian, por su infinita paciencia y por haberme enseñado todo lo que he aprendido además de ser un gran compañero.

A mis compañeros de Universidad. A Julián, Diego Sierra y Diego Lucero por todas las experiencias vividas y buenos momentos juntos y con los que he afrontado muchos retos; junto con Lucía, Robi y Marga.

A Carlos, por su incondicional ayuda y apoyo ante todo tipo de circunstancias además de todos los buenos momentos.

A mis amigos de toda la vida, los "samberos". A mi amigo Dani, por animarme a tomar la decisión correcta en el momento oportuno.

Por último a Pablo, por haber sido mi tutor de este proyecto y haberme brindado esta posibilidad.

Cualquier tecnología suficientemente avanzada es indistinguible de la magia.
Arthur C. Clarke (1917-2008)

Resumen

IMDEA Networks Institute es un instituto de investigación creado por el Gobierno de la Comunidad de Madrid en el que se realiza investigación científica de vanguardia en todas las áreas de redes. Por ello, cada año se reciben cientos de solicitudes de investigadores interesados en unirse al equipo, solicitudes que aumentan año a año.

Inicialmente, el instituto optó por la contratación de una empresa externa encargada de proveer un formulario para la tramitación de dichas solicitudes. Las solicitudes eran enviadas a nuestros servidores diariamente en formato *XML* y cargadas en una aplicación *php* difícil de mantener y obsoleta que, dado el alto número de solicitudes, no se adaptaba a los requerimientos actuales.

Por ello, debido a este aumento y al constante crecimiento del instituto nace el proyecto ARP *(Application Recruitment Procedure)*, con la intención de centralizar el procesamiento de todas las solicitudes de forma interna.

Para el desarrollo de la herramienta se ha decidido utilizar el framework *Ruby On Rails*, y el proyecto se ha dividido en dos partes o interfaces. La primera de ellas es un nuevo formulario web, más intuitivo, dinámico y de carácter público.

La segunda de ellas consiste en una herramienta interna para el procesamiento de dichas solicitudes en detrimento de la anterior. En ella el equipo de administración se encargará de procesar todas las solicitudes que se vayan recibiendo durante el periodo de contratación.

En este Trabajo de Fin de Grado se describirá el proceso de realización de la interfaz primera o formulario web, herramientas y lenguajes utilizados en el proceso y las pruebas realizadas para testear su correcto funcionamiento.

Palabras clave: Ruby On Rails, formulario web, base de datos, interfaz.

Abstract

IMDEA Networks Institute is a research institute created by the Madrid Regional Government in which cutting-edge scientific research is carried out in all areas of networking. Therefore, each year hundreds of requests for researchers interested in joining the team are received, increasing every year.

Initially, the institute chose hiring an external company to provide a form for the processing of such applications. Applications were sent to our servers daily in *XML* format and loaded into a *php* application difficult to maintain and outdated that given the high number of applications, it was not adapted to the current requirements.

Therefore, due to this increase and the continued growth of the institute, borns ARP project *(Application Recruitment Procedure)*, with the intention to centralize the processing of all requests internally.

For the development of the tool we have decided to use the *Ruby On Rails* framework, and the project has been divided into two parts or interfaces. The first is a new, more intuitive, dynamic and public web form.

The second of these is an internal tool for processing such requests. In it, the management team will process all requests that are received during the registration period.

In this Final Project Work will be described the process of realization of the first interface or web form, languages and tools used in the process and tests done to check that everything works properly.

Keywords: Ruby On Rails, formulario web, base de datos, interfaz.

Índice General

Resumen VII

Abstract IX

Índice General XI

Lista de Figuras XV

Glosario XVII

I Introducción 1

1. Introducción 3
1.1. Introducción . . . 3
1.2. Objetivos . . . 3
1.3. Fases del desarrollo . . . 4
1.4. Estructura de la memoria . . . 4

II Estado del Arte 7

2. Ruby On Rails para el desarrollo Web 9
2.1. Introducción . . . 9
2.2. Lenguaje Ruby . . . 10
2.3. Framework de desarrollo . . . 10
2.3.1. Ruby On Rails . . . 10
2.4. Metología de Implementación: MVC . . . 11
2.4.1. Patrón MVC en Ruby on Rails . . . 12
2.5. Desarrollo Web Ágil . . . 13

III Descripción del trabajo realizado 15

3. Herramientas empleadas 17
3.1. Introducción . . . 17
3.2. Gestión de código fuente . . . 17
3.3. Entorno de desarrollo . . . 17
3.4. Complementos y gemas . . . 18

3.5. Conjunto de pruebas para la aplicación 19

4. Estructura de ficheros de la Aplicación **21**
4.1. Introducción 21
4.2. Estructura de ficheros y directorios 21

5. Diseño de los modelos **23**
5.1. Introducción 23
5.2. Objetivos 23
5.3. Los modelos y sus relaciones 23
5.4. Generación de una clase modelo 26
5.4.1. Ficheros de migración 26
5.4.2. Ficheros de la clase Modelo 27
5.4.2.1. Clase User 27
5.4.2.2. Clase Candidate 27
5.4.2.3. Clase Call 28
5.4.2.4. Clase Application 28
5.4.2.5. Resto de clases 28
5.5. Validaciones en el modelo 29
5.5.0.6. Métodos suplementarios 30

6. Diseño de los controladores **31**
6.1. Introducción 31
6.2. Objetivos 31
6.3. Controladores y acciones permitidas 31
6.3.1. Clases con un único recurso 32
6.3.2. Clases con varias instancias 32
6.3.3. CRUD y resourceful Routes 33
6.3.3.1. Resources y Singular Resources 33
6.3.4. Acciones del controlador 34
6.3.5. Control de acceso 35

7. Diseño de las vistas **37**
7.1. Introducción 37
7.2. Objetivos 37
7.3. Vistas principales 37
7.3.1. Sign In y Sign Up 38
7.3.2. Dashboard 39
7.3.3. Candidate 40
7.3.4. Open Positions 41
7.3.5. Application 42
7.3.6. Academic Qualifications, Professional Experience y References 43
7.3.7. Research Interests and statement of purpose 44
7.3.8. Standarized Scores 45
7.3.9. Documents 46
7.3.10. Other relevant information 46
7.3.11. Acción Submit 47

7.4. Layouts . . . 49
7.5. Generación de PDF . . . 49

IV Conclusiones y trabajos futuros **51**

8. Conclusiones y trabajos futuros **53**
8.1. Conclusiones . . . 53
8.2. Trabajos futuros . . . 54

Bibliografía **55**

V Anexos **57**

A. Planificación y presupuesto **59**
A.1. Descomposición en tareas . . . 59
A.2. Planificación con el diagrama de fases de ejecución detallado . . . 66
A.3. Recursos . . . 68
A.4. Presupuesto de Proyecto . . . 68
A.5. Marco regulador . . . 68

B. Configuración del entorno de desarrollo **71**
B.1. Introducción . . . 71
B.2. Características del equipo de desarrollo . . . 71
B.3. Configuración del framework Ruby on Rails . . . 71
B.3.1. Instalar Homebrew . . . 72
B.3.2. Instalar Ruby . . . 72
B.3.3. Instalar Rails . . . 72
B.3.4. Configurar Git . . . 73
B.3.4.1. Instalar Git . . . 73
B.3.4.2. Añadir las llaves SSH al repositorio . . . 73
B.3.4.3. Descargamos la estructura . . . 74
B.3.4.4. Creamos la rama de desarrollo . . . 74

C. Configuración de la gema Devise **75**
C.1. Introducción . . . 75
C.2. ¿Qué es devise? . . . 75
C.3. Instalación . . . 76
C.3.1. Modelo . . . 76
C.3.2. Controlador . . . 77
C.3.3. Vista de inicio de sesión . . . 78

Lista de Figuras

2.1. MVC y rails. Fuente: [31] . . . 12
2.2. Fases del modelo de desarrollo en cascada . . . 13
2.3. Modelo de desarrollo en etapas solapadas . . . 14

5.1. Estructura del modelo relacional . . . 25
5.2. Validación de datos requerida antes de rellenar la aplicación . . . 30
5.3. Tabla que muestra el estado de cada parte del formulario . . . 30

7.1. Vista de las acciones Sign in y Sign up . . . 38
7.2. Dashboard o vista principal de la página . . . 39
7.3. Vista Candidate que contiene los datos personales del usuario . . . 40
7.4. Lista de posiciones abiertas a contratación . . . 41
7.5. Vista de "Submit" de la aplicación a la posición . . . 41
7.6. Vista principal de la aplicación . . . 42
7.7. Vista Index de Relevant Profesional Experience . . . 43
7.8. Vista Supervisors mostrada al añadir un nuevo elemento . . . 44
7.9. Vista Standarized Scores referente a conocimientos de la lengua inglesa . . . 45
7.10. Vista Documents donde se muestran los archivos añadidos . . . 46
7.11. Vista Application tras completar todos los campos . . . 47
7.12. Vista Application tras haber enviado el formulario . . . 48
7.13. Ejemplo de documento PDF generado por la aplicación . . . 50

A.1. Diagrama de Gantt con la planificación del proyecto resumida . . . 66
A.2. Diagrama de Gantt con la planificación detallada del proyecto . . . 67

B.1. Interfaz web Gitlab donde añadir la llave SSH . . . 74

Glosario

Action Pack Patrón de diseño que divide la respuesta a una solicitud web en una parte del controlador (lógica) y una parte de la vista (plantilla).

Active record Patrón de diseño que se puede encontrar en software que accede a bases de datos. Se basa en que una instancia de un objeto está enlazada a una fila de una tabla en la base de datos. Cuando se crea un objeto, se añade una nueva fila al guardar. Cuando un objeto se modifica, su fila también lo hace.

Framework Estructura definida para el desarrollo y/o implementación de una aplicación modular.

Patrón Modelo Vista Controlador (MVC) Es un estilo de arquitectura software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.

COC (Convenction over Configuration) Patrón de diseño que busca disminuir la cantidad de decisiones que un desarrollador debe tomar, siguiendo un camino estándar sin necesariamente perder flexibilidad y donde el desarrollador sólo necesita especificar las partes de la aplicación que no sean convencionales.

DRY (Don't Repeat Yourself) Filosofía que considera innecesario repetir código si éste está bien diseñado y refactorizado.

Ruby Lenguaje de programación orientado a objetos de script con características propias de lenguajes funcionales.

Ruby On Rails (RoR) Framework de desarrollo rápido de aplicaciones web basado en el lenguaje de programación

GIT Sistema de control de versiones.

SQLite3 Sistema de gestión de base de datos de software libre.

JavaScript Lenguaje de script que se ejecuta en el lado cliente.

Desarrollo web ágil iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración dentre desarrolladores organizados correctamente.

CRUD acrónimo de Crear, Obtener, Actualizar y Borrar (del original en inglés: Create, Read, Update and Delete). Se usa para referirse a las funciones básicas de la base de datos

Gema plugins y/o códigos añadidos a nuestro proyecto Ruby on Rails, que nos permite añadir nuevas funcionalidades.

Rspec conjunto de pruebas y validaciones para el desarrollo de la aplicación

Parte I

Introducción

Capítulo 1

Introducción

1.1. Introducción

El instituto IMDEA Networks se encuentra en fase de crecimiento y en continua búsqueda de nuevos candidatos. La demanda de solicitudes crece año a año y la herramienta que se utilizaba no cumplía con los requisitos actuales, por lo que se optó por desarrollar una herramienta desde cero para gestionar dichas solicitudes.

En este Trabajo de Fin de Grado se describe el desarrollo de la nueva solución, más intuitiva, escalable, sencilla de mantener y con un uso real. Este proyecto consta de dos aplicaciones: la aplicación para recibir las solicitudes y la aplicación para su tramitación. Este trabajo se centrará en la primera de ellas.

1.2. Objetivos

Los objetivos perseguidos en este trabajo de fin de grado son:

- Diseño, desarrollo y despliegue de una aplicación web para recibir solicitudes que sincronice con otra aplicación de gestión de dichas solicitudes.
- Empleo de herramientas de software libre por la naturaleza de la licencia abierta
- Crear una herramienta sencilla de instalar, mantener y escalable para futuras implementaciones.
- Conseguir desarrollar una herramienta robusta y estable gracias a las pruebas *rspec*.
- Ajustarse lo máximo posible a las especificaciones de la organización en cuanto a diseño y requerimientos de la aplicación, proponiendo soluciones alternativas en el caso de no poder implementar diversas funcionalidades.

1.3. Fases del desarrollo

El Trabajo Fin de Grado se dividió en las fases de desarrollo siguientes:

- **Análisis y requerimientos iniciales**: estudio del proyecto propuesto por la organización y análisis del mismo, estableciendo las herramientas a utilizar.
- **Diseño**: diseño y desarrollo de la aplicación haciendo uso del *MVC* (modelo-vista-controlador) bajo el framework *Ruby on Rails*.
- **Implementación**: integración del diseño con la base de datos, sistema de autenticación, procesado de documentos y correos automáticos.
- **Testing**: pruebas de código, validaciones, de seguridad y corrección de errores.
- **Presentación de la primera beta**: se presentó una primera versión funcional de la herramienta, y se realizaron los cambios propuestos durante la misma.
- **Sincronización de aplicaciones**: se comprobó que ambas aplicaciones sincronizaban de forma correcta y se realizaron los ajustes pertinentes.
- **Desarrollo**: se puso la herramienta en funcionamiento, dotándola de acceso público.

1.4. Estructura de la memoria

La memoria se divide en varias partes, que a su vez se dividen en distintos capítulos. El contenido de cada parte y capítulo se resume a continuación:

1. **Primera parte: Introducción**. En esta parte se explican los objetivos, las fases del trabajo y la estructura de la memoria. Sólo contiene un capítulo:
   - Capítulo 1. Introducción
2. **Segunda parte: Estado del arte**. Se explican los diferentes retos que motivaron al desarrollo de la aplicación y de las herramientas empleadas
   - Capítulo 2. *Ruby On Rails* para el desarrollo Web
3. **Tercera parte: Descripción del trabajo realizado**. En esta parte se explica el trabajo realizado para la puesta en funcionamiento de la aplicación Web.
   - Capítulo 3. Herramientas empleadas.
   - Capítulo 4. Estructura de ficheros de la Aplicación.
   - Capítulo 5. Diseño de los *modelos*.
   - Capítulo 6. Diseño de los *controladores*.
   - Capítulo 7. Diseño de las *vistas*.

4. **Cuarta parte: Conclusiones y trabajos futuros**. En esta parte se explican las conclusiones obtenidas del trabajo y futuras implementaciones para ampliar la herramienta actual. Sólo contiene un capítulo:

   - Capítulo 6. Conclusiones y trabajos futuros

5. **Quinta parte: Anexos**. En esta última parte se amplía la información de algunas partes del Trabajo Fin de Grado:

   - Apéndice A: Planificación de tareas y presupuesto. En este anexo se describen las tareas del proyecto y los costes de estas.
   - Apéndice B: Configuración del entorno de desarrollo. Se describe el proceso de instalación y puesta a punto del entorno de desarrollo.
   - Apéndice C: Configuración de la gema *Devise*. Se presenta la configuración detallada del módulo o gema que dota de autentificación a la aplicación.

Parte II

Estado del Arte

Capítulo 2

Ruby On Rails para el desarrollo Web

2.1. Introducción

Presentada y analizada la propuesta del proyecto por parte de la organización, comenzamos a estudiar cómo desarrollarla. En vista de los requerimientos, decidimos que lo más óptimo para desarrollar la aplicación sería hacer uso de un lenguaje que cumpliese las siguientes características:

- Lenguaje de alto nivel
- Orientado a objetos
- Compatible con el patrón de diseño *Modelo-Vista-Controlador*

Aplicadas estas restricciones, las posiblidades se redujeron drásticamente. Las alternativas más atractivas tras el filtro fueron *Java*[3], *Ruby*[26]. y *Python*[1]. De cara a desarrollar una aplicación web, buscamos que se tratara de un desarrollo dinámico, sencillo de probar y rápido. En ese preciso instante de descartó *en*Java. Resulta incómodo que, para cada cambio por mínimo que sea en la aplicación, haya que compilar el código.

Tras ello, la decisión entre *Ruby* y *Python* resultó más complicada. Python es un lenguaje más extendido y popular que Ruby. Es un lenguaje sencillo de aprender y sitios de la envergadura de *Google* o *Youtube* se han desarrollado con él. Pese a ello, la decisión final fue escoger *Ruby*. Los principales motivos por los que decidimos escogerlo como lenguaje a utilizar en este proyecto fueron:

- Documentación más clara y sencilla
- Multitud de herramientas y complementos para desarrollar formularios
- Posee mejores características de seguridad

Por último, otra de las razones de peso que nos llevó a tomar la decisión de elegir Ruby fue la existencia del framework de desarollo *Ruby On Rails*[25], del que hablaremos más adelante.

2.2. Lenguaje Ruby

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos. Es considerado una mezcla de lenguajes como *Perl, Smalltalk, Eiffel, Ada, y Lisp*. Las ventajas que *Ruby* nos ofrece son varias:

- Es software libre. Está liberado bajo licencia GPL [4].
- Es cómodo de usar. El código resulta claro y legible, por lo que lo hace ideal para trabajar en grupo.
- Es multiplataforma. Puede usarse tanto en Windows, Linux o Mac OS.
- Es el lenguaje que utiliza el ya mencionado framework *Ruby On Rails*, en el cual profundizaremos a continuación.

2.3. Framework de desarrollo

La diferencia entre crear una aplicación web con o sin la ayuda de un *framework* de desarrollo es enorme. Un buen *framework* no sólo ahorra tiempo sino que ayuda a que el resultado sea más mantenible y cumpla con una serie de mejores prácticas a la hora de programar la aplicación.

2.3.1. Ruby On Rails

Ruby on Rails(también llamado simplemente *Rails* o *RoR*) nace como *framework* en 2004, ofreciendo una estructura definida para el desarrollo de una aplicación modular sobre el lenguaje *Ruby*.

Rails ahorra una cantidad de tiempo enorme a la hora de desarrollar una aplicación web en comparación con otros *frameworks*. Parte de ese ahorro de tiempo se debe a que *Ruby on Rails* asume que el programador va implementar la aplicación de una determinada forma y hace parte del trabajo por el desarrollador.

Además, *Rails*, promueve una serie de buenas prácticas a la hora de programar, como son las siguientes:

- No repetirse a si mismo (en inglés "Don't Repeat Yourself" o *DRY*). Hace alusión al hecho de que en programación nunca debe haber código duplicado.
- El código debe ser legible.
- Hacer uso de un patrón de diseño, como es el *Modelo-Vista-Controlador*.
- Hacer una aplicación robusta y bien testeada.

La decisión de escoger *Ruby On Rails* como *framework* para el desarrollo web se tomó por diversas razones. Es uno de los *frameworks* con más proyección de futuro. *Ruby On Rails* nos ofrece numerosas ventajas:

- Software libre que funciona con multitud de distintas *bases de datos*, lo que inicialmente reduce los costes sin sacrificar velocidad, seguridad o rendimiento. Funciona bajo licencia MIT [5].
- Gran flexibilidad de cara a incrementar funcionalidades según de los requerimientos.
- Se fundamenta en el *desarrollo web ágil*.
- Permite añadir *gemas* a nuestro proyecto. Las *gemas* son plug-ins y/o librerías añadidas que dotan a la aplicación de nuevas funcionalidades, como por ejemplo el login de usuarios.
- La comunidad de desarrollo de *Ruby* ofrece infinidad de recursos y se encuentra en constante desarrollo.

2.4. Metología de Implementación: MVC

El diseño Modelo Vista Controlador *(MVC)* propone dividir las aplicaciones en tres tipos de componentes: *modelos*, *vistas* y *controladores*.

El *modelo* es responsable de mantener el estado de la aplicación. En el caso de nuestra aplicación, el uso principal que tiene es establecer la relación entre los distintos objetos de la aplicación y encargarse de realizar las validaciones pertinentes para proporcionar fiabilidad.

La *vista* es responsable de generar la interfaz de usuario, normalmente en función de los datos del *modelo*. Pese a ello, la *vista* nunca se encargará de manejar por ella misma los datos que se reciben. Una vez la vista es mostrada, su trabajo ha terminado.

El *controlador* es quien se encarga de manejar y complementar los dos anteriores. Recibe los datos (normalmente por parte del usuario), interactúa con el *modelo* y lo muestra de forma correcta en la *vista* al usuario.

Cabe destacar el componente *routing*, clave en *Rails*. Éste recibe una petición e inmediatamente la separa. La petición contiene un método y una ruta determinados. Hablaremos de ellos más adelante.

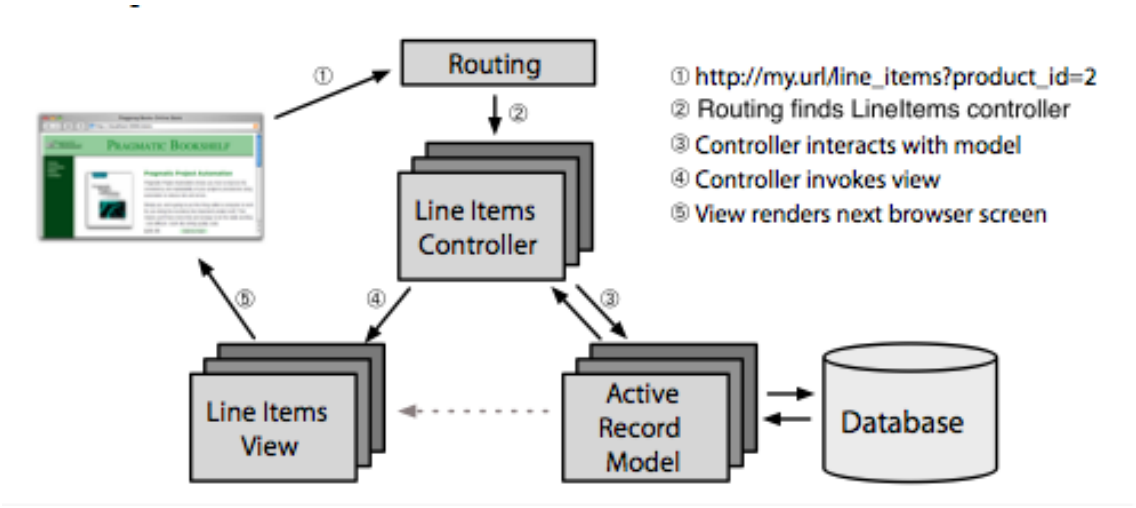


Figura 2.1: MVC y rails. Fuente: [31]

2.4.1. Patrón MVC en Ruby on Rails

Tomaremos como ejemplo la figura anterior.

1. El navegador envía una petición (*http*)
2. El sistema de enrutado decide a qué *en*controlador enviarla
3. El *controlador* interactúa con el *modelo*
4. El *controlador* invoca la *vista* (respuesta *http*)
5. El navegador dibuja la *vista* en la pantalla

Para que esto funcione, *Rails* hace uso de dos componentes esenciales:

1. **Action Pack**. Se considera "el corazón de *Rails*". Está compuesto por:

   Action Controller + Sistema de routing. Una vez routing determina el *controlador* a usar para procesar la petición, el *controlador* se encarga de dar sentido a la solicitud y produce la salida adecuada. Gracias a las convenciones, esta tarea se realiza de forma simple para el programador.

   Action View. Las plantillas Action View utilizan codigo *Ruby* embebido mezclado con HTML. Proporciona clases de ayuda para el comportamiento de formularios, fechas y cadenas de texto, lo que evita en muchas ocasiones código repetitivo.

2. **Active Record**. Facilita la creación y uso de objetos en el *modelo* cuyos datos requieren de ser almacenados. Se trata de un mapeo de objetos relacionales.

2.5. Desarrollo Web Ágil

Las metodologías o *frameworks* de desarrollo ágil surgen como una respuesta a métodos clásicos de gestionar proyectos como el modelo en cascada.

El desarrollo en cascada fue introducido en un artículo escrito por Winston Royce en 1970 [30], principalmente orientado a proyectos gubernamentales. Siguiendo este modelo, el desarrollo de un proyecto se divide en varias fases de varios meses de duración, que deben sucederse en orden estricto y no dan un producto terminado hasta el final del ciclo completo. Un esquema de desarrollo en cascada se observa en la figura 2.2:

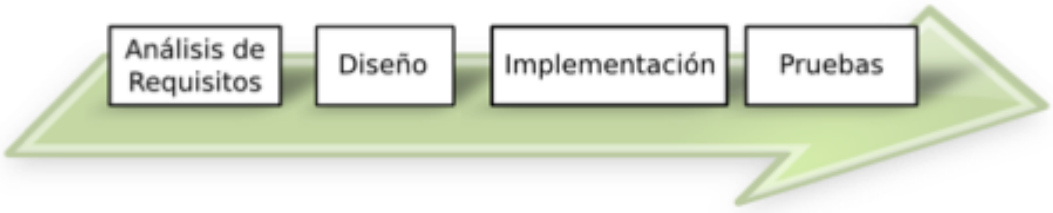


Figura 2.2: Fases del modelo de desarrollo en cascada

Este método tiene su origen en la ingeniería clásica, como la construcción de edificios, puentes, etc. En la ingeniería clásica, los costes de realizar cambios en el proyecto crecen de forma exponencial conforme se avanza en el mismo. Resulta impensable cambiar la estructura de un edificio cuando ya se ha empezado a construir. El método en cascada minimiza los costes totales de desarrollo colocando al inicio del mismo unos análisis y diseño exhaustivos que no deben ser alterados.

Sin embargo, este método no parece adaptarse bien al desarrollo de tecnologías modernas que están sujetas a constantes cambios y son mucho más imprevisibles en su diseño. Estudios sobre el éxito de los proyectos de software realizado en 1994 [32] demuestran que menos de un 16 % de los mismos son terminados a tiempo y dentro del presupuesto previsto. El principal punto de conflicto es la respuesta al cambio. El método en cascada asume que es posible capturar todos los requisitos de un proyecto al inicio del mismo con precisión. Sin embargo, la experiencia ha demostrado que es inevitable dejar cosas atrás. Además, en el ámbito del software es muy común que el cliente debe ver el producto para saber qué es lo que realmente quiere.

A la vista de estos problemas, a finales de los 70, compañías especializadas en el desarrollo de productos lideradas por Toyota, Honda, Fujitsu, 3M, HP, Canon y NEC sustituyeron el método tradicional por otro en el que las distintas fases se solapan las unas a las otras. Véase figura 2.3.

Figura 2.3: Modelo de desarrollo en etapas solapadas

Los resultados fueron un descenso dramático en el tiempo y coste del desarrollo de sus productos. Estos nuevos métodos evolucionaron y dieron lugar a métodos conocidos como *lean development* y *just in time manufacturing*. A partir de los 90 son muchas las empresas que han adoptado un método de desarrollo iterativo e incremental.

Irónicamente, el desarrollo de software ha estado fuertemente rezagado en este aspecto y no ha empezado a adoptar *metodologías ágiles* como Scrum hasta finales de los 90. Sin embargo, el éxito ha sido tremendo y ahora es usado por grandes compañías como Google, Yahoo, Microsoft, IBM, Oracle, Adobe, General Electric, Siemens, BellSouth, Sony/Ericson, Accenture, Philips, Barclays Global Investors, Nokia, etc. [34]

Como resumen de los valores que el desarrollo ágil promueve, en 2001, un grupo de expertos se reunieron y redactaron lo que hoy se conoce como *Manifiesto Ágil* [27]. En el que afirman lo siguiente:

Estamos descubriendo mejores formas de desarrollar software haciéndolo y ayudando a otros a hacerlo. A través de este trabajo hemos llegado a valorar:

- *Individuos e interacciones por encima de procesos y herramientas.*
- *Software que funciona por encima de documentación extensa.*
- *Colaboración con el cliente por encima de negociación de contratos.*
- *Responder a cambios frente a seguir un plan.*

A pesar de que valoramos los elementos de la derecha, valoramos más los que se encuentran a la izquierda.

Parte III

Descripción del trabajo realizado

Capítulo 3

Herramientas empleadas

3.1. Introducción

En este capitulo se recopilarán las herramientas usadas durante el desarrollo de la aplicación y una breve explicación de por que se han escogido tales herramientas y lo que nos aportan.

3.2. Gestión de código fuente

Una sistema de gestión de código fuente permite tener versiones distintas de un proyecto, mostrando todos los cambios que se han hecho a lo largo del tiempo y permitiendo dar marcha atrás si es necesario y deshacer los cambios. Ya únicamente la habilidad de poder comparar dos versiones y revertir los cambios lo dota de un valor incalculable en cualquier proyecto de desarrollo de software.

La elección del sistema para gestionar el código fuente fue fácil. Desde hace un tiempo que la que comunidad *Rails* hace uso de *Git*[23] para solucionar este problema y parece una decisión de lo más acertada.

Git facilita enormemente todas las funciones básicas que se esperan de un buen *Source Code Management* tales como permitir que varias personas trabajen simultáneamente en el mismo proyecto o llevar varias ramas de desarrollo del mismo. También tiene la característica de que es descentralizado, algo que se agradece y además permite trabajar en local sin necesidad de tediosas configuraciones de un servidor.

3.3. Entorno de desarrollo

Con respecto al entorno de desarrollo, se ha optado por el uso del editor de texto *Sublime Text 2*[24]. Como editor de texto base no dista mucho del resto, pero sí es posible añadir multitud de plug-ins que de cara al desarrollo hacen todo más fácil.

También Se ha hecho uso de la consola de *Rails*. Gracias a ella podemos comprobar el funcionamiento de métodos de forma sencilla sin necesidad de abrir el navegador.

3.4. Complementos y gemas

Además de estas herramientas básicas, *Ruby on Rails* dispone de un sistema de librerías o *Gemas*. Estas *gemas*, añadidas a nuestro proyecto, nos permiten añadir funcionalidades adicionales a nuestra aplicación de forma sencilla.

En este apartado vamos a nombrar las principales *gemas* y complementos que hemos usado en la aplicación del formulario:

- **rails**[17]. Se considera la *gema* principal. Todo proyecto en *Rails* debe tener esta *gema*.
- **sqlite3**[2]. Permite a la aplicación interactuar con bases de datos *SQLite*. Es ideal para el entorno de desarrollo por su bajo consumo de recursos y fácil uso.
- **uglifier**[20]. Permite el uso de herramientas *JavaScript* en código *Ruby*
- **jquery-rails**[15]. Permite el uso de *JQuery* en nuestra aplicación.
- **jbuilder**[13]. Permite declarar mediante codigo *Ruby* estructuras *JSON* de forma sencilla.
- **bootstrap**[9]. Conjunto de herramientas HTML, CSS y JavaScript que permiten desarrollar código de forma sencilla, intuitiva y visualmente atractiva.
- **devise**[12]. Permite dotar de autentificación a la aplicación. Se explicará más adelante.
- **rolify**[18]. Permite definir roles entre los usuarios de forma sencilla. Perfectamente integrada con la *gema devise*.
- **simple_form**[19]. Permite la creación de formularios de forma sencilla e intuitiva. Perfectamente integrada con *bootstrap*.
- **country_select**[11]. Helper para *simple_form* que nos permite seleccionar un país de forma sencilla.
- **paperclip**[16]. Permite la subida/descarga de documentos. Será la base sobre la que actuará la gema *jquery-fileupload-rails*.
- **jquery-fileupload-rails**[14]. Conjunto de funcionalidades para la subida/descarga de ficheros. Proporciona una interfaz sencilla y visual.
- **breadcrumbs_on_rails**[10]. Permite añadir *breadcrumbs* de forma sencilla a una aplicación *Rails*.
- **wkhtmltopdf-binary**[8]. Libería que permite renderizar archivos HTML en archivos PDF. Necesaria para la generación de PDF's mediante la gema *wicked_pdf*.
- **wicked_pdf**[21]. Hace uso de la librería anterior para generar PDF's a través de vistas HTML. Añade numerosas opciones para la creación de PDF's.

3.5. Conjunto de pruebas para la aplicación

En el desarrollo con *Rails* se hace mucho hincapié en que las aplicaciones deben tener tests. Existen múltiples herramientas para escribir dichos tests. En nuestro caso hemos escogido *Rspec*, dado que produce unos tests especialmente fáciles de leer y entender.

Para ello nos hemos apoyado en el libro "Everyday Rails Testing with RSpec: A practical approach to test-driven development".[33]

Capítulo 4

Estructura de ficheros de la Aplicación

4.1. Introducción

Todas las aplicaciones *Rails* poseen la misma estructura de ficheros siguiendo el paradigma *CoC (Convention Over Configuration)*. A continuación se especifican los directorios principales y ficheros más importantes de la aplicación.

4.2. Estructura de ficheros y directorios

La estructura de ficheros de la aplicación es la siguiente:

- **app/** Contiene la aplicación en sí. La inmensa mayoría del código escrito para la aplicación, se encuentra aquí.
- **app/models/** Contiene un fichero por cada *modelo* que contiene la aplicación.
- **app/controllers/** Contiene un fichero por cada *controlador* que contiene la aplicación.
- **app/views/** Contiene un fichero por cada *vista* que contiene la aplicación.
- **app/helpers/** Contiene métodos que pueden ser usados en las *vistas*.
- **config/** Contiene todos los ficheros relativos a la configuración de la aplicación.
- **config/locales/database.yml** Contiene la información necesaria para que la aplicación sea capaz de conectarse a la base de datos.
- **config/locales/routes.rb** Contiene toda la configuración de rutas de las *vistas* y *controladores*.
- **db/** Contiene toda la información relativa a la base de datos.
- **db/migrate/** Contiene todos los ficheros de migración relativos a los modelos.

- **doc**/ Documentación de la aplicación que puede generarse automáticamente a través de los comentarios en el código.
- **lib**/ Clases usadas por el resto de la aplicación.
- **LICENSE** Licencia bajo la que se distribuye la aplicación.
- **log**/ Alberga los logs de la aplicación.
- **public**/ Contiene aquellos ficheros que son accesibles directamente por los usuarios tales como imágenes u hojas de estilo.
- **Rakefile** Contiene tareas automatizadas.
- **README** Fichero de texto que contiene la explicación sobre la instalación y uso de la aplicación
- **script**/ Contiene scripts para varias tareas como lanzar el servidor.
- **spec**/ Contiene todo lo relativo a los tests.
- **tmp**/ Contiene ficheros temporales.
- **vendor**/ Contiene los plugins y otros ficheros que no pertenecen a la aplicación como tal.

Capítulo 5

Diseño de los modelos

5.1. Introducción

A lo largo de este capítulo vamos a tratar sobre qué clases que componen la capa de *modelo* de la aplicación y como se han implementado.

5.2. Objetivos

El objetivo es mostrar los datos que la aplicación va a almacenar y la relación que guardan las tablas entre sí. Puesto que la aplicación cuenta con más de 15 tablas, nos centraremos en explicar las principales clases y la función de cada una.

5.3. Los modelos y sus relaciones

Como se explicó en el capítulo 2, los *modelos* hacen uso de la clase *Active Record* para hacer un mapeo relacional entre las clases y las tablas de la base de datos. Las principales clases son las siguientes:

- **User**. Representa cada uno de los usuarios del sistema. Contiene todos los datos referentes a la sesión, como correo electrónico, password, etc.
- **Candidate**. Contiene los datos personales del usuario y datos de contacto, y se relaciona de forma directa con la clase *User*.
- **Application**. La clase *Application* aunará todos los datos relacionados con el formulario. Depende directamente de *Candidate* y *Call*. Siempre que haya una call abierta, el usuario podrá aplicar. Por cada *Call* que aplique, se generará un nuevo *Application* (habrá tantos *Application* como *Calls* el usuario haya aplicado).
- **Call**. Se generará cada vez que la organización inicie un proceso de contratación. Contendrá los datos de la posición, descripción, inicio y fin de la misma.

El resto de clases enlazan con *Application*, y el conjunto forma todos los datos que la organización requiere en el formulario. El *modelo* relacional se muestra en la siguiente figura:

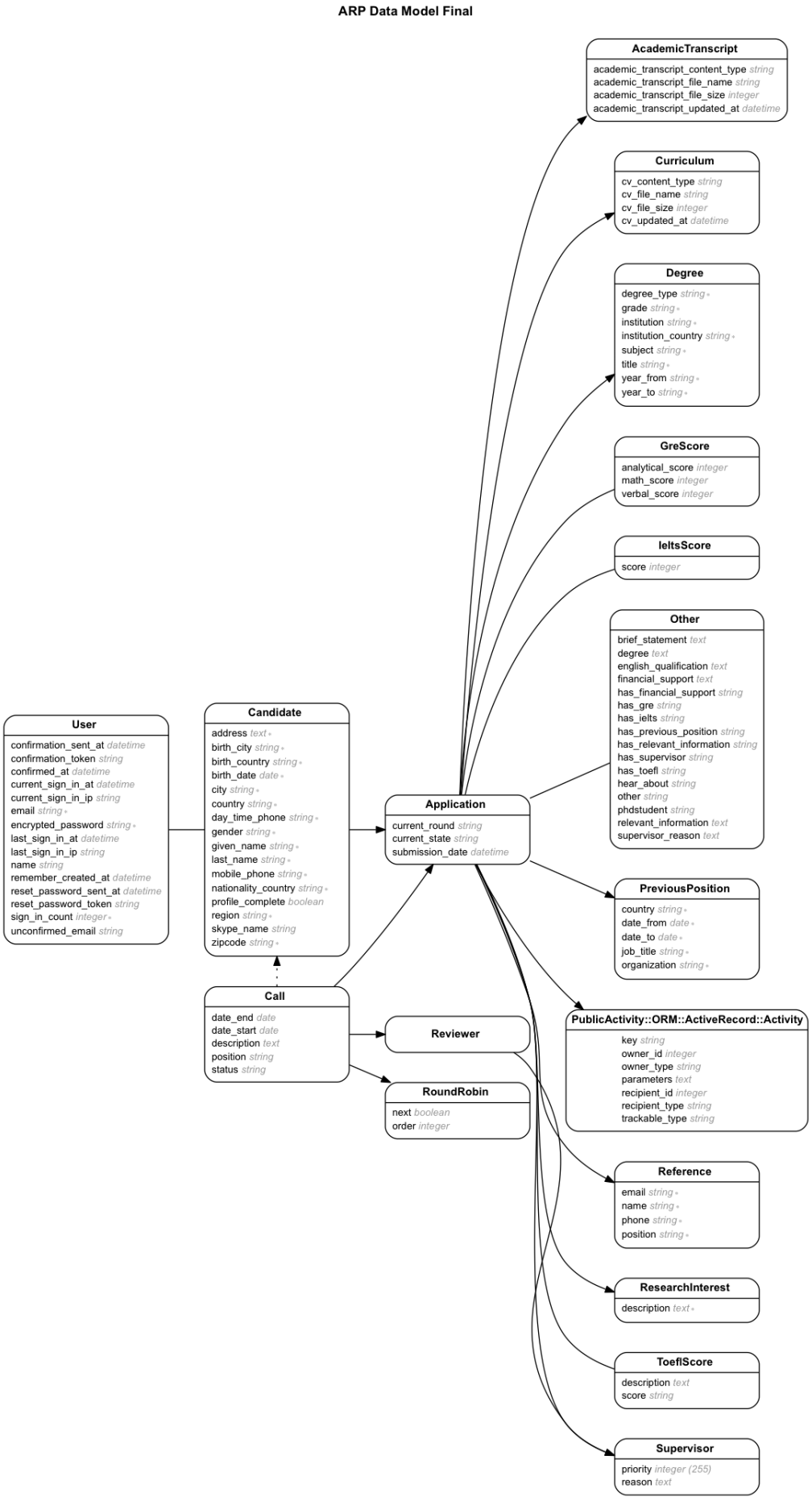


Figura 5.1: Estructura del modelo relacional

5.4. Generación de una clase modelo

Rails provee de un generador de código para automatizar la creación de una clase *modelo*. Así por ejemplo podríamos crear la clase *Call* con el siguiente comando:

```
$ rails generate model Call parameter:data_type
```

Dicho comando genera dos ficheros esenciales para definir una clase *modelo*:

- Un fichero de *migración*, que incluye la información necesaria para crear las tablas y columnas correspondientes dentro de la base de datos.
- Un fichero para definir una *clase*, dentro del cuál estará definida la clase en sí, con sus métodos, etc.

Además de estos dos, también genera ficheros relativos a las pruebas de la aplicación que por simplicidad vamos a ignorar.

5.4.1. Ficheros de migración

Contienen la información necesaria para crear las tablas y columnas requeridas para guardar los objetos de dicha clase en la base de datos. Se guardan bajo el directorio `db/migrate/` y se ejecutan por orden con el comando: `rake db:migrate`

A modo de ejemplo, el fichero de migración que se generará para la clase *Candidate* de nuestra aplicación es el siguiente:

```
class CreateCandidates < ActiveRecord::Migration
  def change
    create_table :candidates do |t|
        t.string :given_name
        t.string :last_name
        t.date  :birth_date
        t.string :birth_city
        t.string :birth_country
        t.string :gender
        t.string :nationality
        t.text :address
        t.string :city
        t.string :region
        t.string :zipcode
        t.string :country
        t.string :day_time_phone
        t.string :fax_number
        t.string :mobile_phone
        t.string :skype_name
        t.timestamps
    end
  end
end
```

Además hacemos uso del método `timestamps`. Dicho método creará las columnas `created_at` y `updated_at` que almacenarán las fechas de creación y actualización respectivamente de forma automática. *Rails* añade por su cuenta una columna más llamada `id`, de tipo entero, que servirá de identificador único para cada objeto de la clase *Candidate*.

5.4.2. Ficheros de la clase Modelo

Rails hace que las clases que declaran un *modelo* sean mucho más simples que en otros *frameworks*. Para ello, los *modelos* heredan de la clase `ActiveRecord::Base`, que es capaz de generar determinados métodos de forma dinámica para ahorrarnos mucho trabajo.

Siguiendo nuestra aplicación, tomemos de ejemplo la clase *Application*. *Rails* asume que cada objeto de la clase *Application* ocupará una fila dentro de la tabla applications y que por cada columna de esa tabla, los objetos de la clase *Application* tendrá un atributo con ese nombre.

Así, establecer las relaciones entre las tablas es muy sencillo. Haremos uso de los métodos `has_one` y `has_many` para las relaciones "hacia abajo", y del método `belongs_to` para determinar a quien pertenece. Una vez definido y de forma dinámica, *Rails* generará un conjunto de métodos que nos permitirán acceder a los atributos de cada una de las tablas.

Las definición de relaciones entre las principales tablas serán las siguientes:

5.4.2.1. Clase User

Se considera la clase principal de la aplicación. Existirá una sola clase por usuario y por cada uno de esos usuarios, un sólo objeto *Candidate*.

Contendrá también todos los parámetros relacionados con la autentificación. Por sencillez y para facilitar la compresión, se han omitido dichos parámetros en el código que se incluye a continuación:

```
class User < ActiveRecord::Base
    has_one :candidate
end
```

5.4.2.2. Clase Candidate

Pertenece a la clase *User* y tendrá tantas aplicaciones como haya hecho el usuario.

```
class Candidate < ActiveRecord::Base
        belongs_to :user
        has_many  :applications
        has_many :calls, :through => :applications
end
```

5.4.2.3. Clase Call

La clase *Call* estará relacionada con todas las aplicaciones de los usuarios, por lo que contendrá las aplicaciones referentes a cada *Call*.

```
class Call < ActiveRecord::Base
    has_many :applications
    has_many :candidates, :through => :applications
end
```

5.4.2.4. Clase Application

Pertenece a la clase *Candidate* y dependerá directamente de la clase *Call*. Cada aplicación irá relacionada con una *Call*, siendo posible una única aplicación por cada una de ellas.

Define también los objetos que cuelgan de cada una de las aplicaciones, y si se trata de objetos singulares o plurales (`has_one` o `has_many`).

```
class Application < ActiveRecord::Base
        belongs_to :call
        belongs_to :candidate

        has_many :degrees
        has_one :gre_score
        has_one :ielts_score
        has_one :other
        has_many :previous_positions
        has_many :references
        has_many :research_interests
        has_one :toefl_score
        has_many :curriculums
        has_many :academic_transcripts
        has_many :supervisors
end
```

5.4.2.5. Resto de clases

El resto de clases de la aplicación pertenecen a *Application*. Por sencillez, omitiremos la parte referente al código de los mismos pues, para el caso que nos atañe, todas harán uso del método `belongs_to` hacia la clase Application.

5.5. Validaciones en el modelo

Las validaciones en el controlador nos permiten asegurarnos de que sólo datos válidos se almacenan en la base de datos. Las validaciones de las que hemos hecho uso en el controlador son las siguientes:

- **De presencia**. Permite asegurarnos de que antes de enviar los datos, se hayan completado los campos solicitados. Así, en cada modelo añadimos la siguiente sentencia:

  ```
  validates :campo_a_completar, presence: true
  ```

- **De tipo**. Nos permite comprobar el tipo de dato que se incluye en el campo solicitado. Por ejemplo, si solicitamos un campo en el que únicamente pueden incluirse números, se hará del siguiente modo:

  ```
  validates :campo_numerico, numericality: { only_integer: true }
  ```

- **De inclusión**. Nos permite validar que el valor del atributo se incluye en un conjunto dado. De hecho, el conjunto puede ser cualquier objeto enumerable. Por ejemplo:

  ```
  validates :has_data, inclusion: { in: %w(Yes No) }
  ```

- **Validaciones en archivos**. Dentro de éstos, realizamos dos tipos de validaciones:

  - **Tipo de contenido**. Comprobamos el content_type del archivo que se va a subir:

    ```
    validates_attachment_content_type :cv,
    :content_type => ["application/pdf", "application/msword", ...]
    ```

  - **Tamaño máximo**. Nos permite limitar el tamaño del fichero a subir:

    ```
    validates_attachment_size :cv,
    :less_than => 2.megabytes
    ```

5.5.0.6. Métodos suplementarios

Además de los métodos que *Rails* ofrece, para el desarrollo de la aplicación hemos necesitado definir otra serie de métodos para añadir comprobaciones personalizadas. Estos métodos se utilizan para definir comportamientos y añadir validaciones extra.

Por ejemplo, en el caso de que un usuario se registre, no podrá rellenar el formulario hasta que antes haya completado su perfil, y esa validación se define en el *modelo* y se ejecuta en el *controlador*.

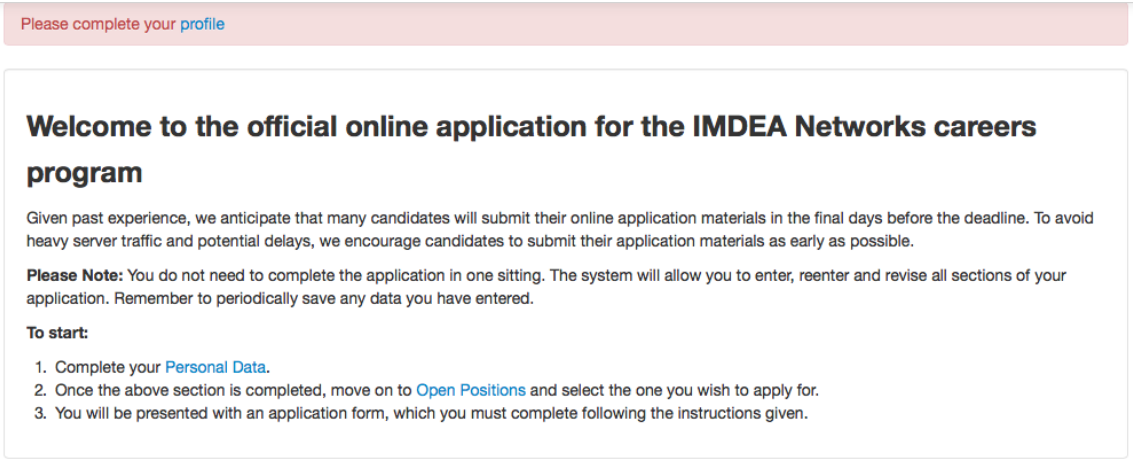


Figura 5.2: Validación de datos requerida antes de rellenar la aplicación

Además, a medida que se vayan completando los datos, en el formulario se indicará si esa parte está completa o necesita de información adicional para ser completada. Que el formulario esté completo es condición necesaria para poder formalizar la aplicación con éxito. Esto se consigue de la misma forma, por medio de métodos suplementarios que comprueban los datos de cada parte del formulario, como se puede observar en la siguiente figura donde se muestra el estado de cada parte (Completed-Incomplete):

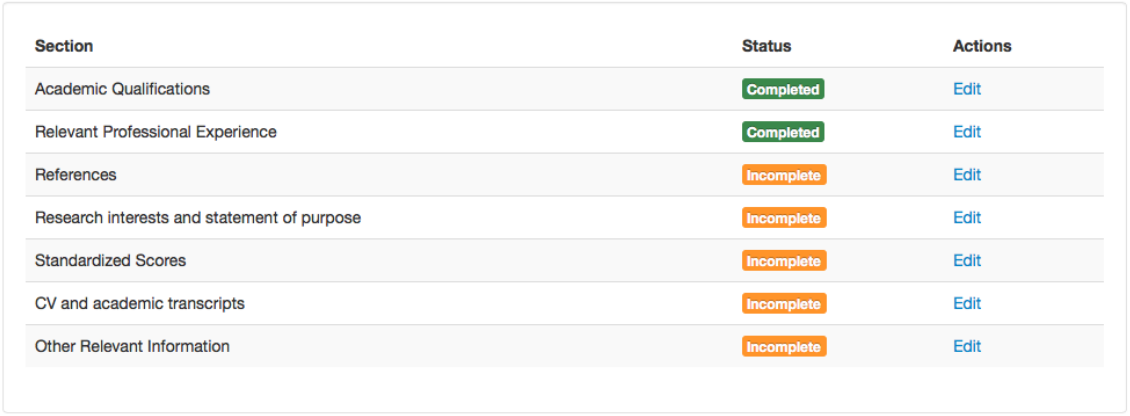


Figura 5.3: Tabla que muestra el estado de cada parte del formulario

Capítulo 6

Diseño de los controladores

6.1. Introducción

En el siguiente capítulo vamos a tratar la capa *controlador*; como interactúa con la capa *modelo* y como invoca a las *vistas* en función de la acción que se vaya a tomar.

6.2. Objetivos

El objetivo de la capa de *controlador* es recibir la petición del usuario, ejecutar la acción requerida haciendo uso de la capa de *modelo* y cargar en memoria aquello que vaya a necesitar la capa de *vista*.

Rails sigue el siguiente patrón. Por cada clase *modelo* (por ejemplo, el *modelo* `Degree`) hay un *controlador* (`DegreesController`). Que posee las acciones necesarias para que el usuario pueda crear, leer, actualizar y eliminar títulos. En Inglés, Create, Read, Update and Delete *(CRUD)*.

No obstante, no se hace uso de todas las acciones en todos los *controladores* y pueden definirse de manera personalizada. Hay clases en las que pueden modificarse los datos, pero no eliminarse; para mantener la consistencia de la base de datos. Por ejemplo, un objeto de la clase *Application*, una vez creado, puede modificarse tantas veces como se quiera pero no se podrá eliminar sin eliminar todos sus datos asociados.

6.3. Controladores y acciones permitidas

Antes de comenzar a explicar cada uno de los *controladores*, vamos a hacer una división. Dado que cada Clase puede tener una o varias instancias de la misma, vamos a separar las clases de las que sólo exista una instancia de las que tienen varias.

6.3.1. Clases con un único recurso

Entendemos por clases con un único recurso por aquellas de las cuales se requiere una sóla instancia. Por ejemplo, para cada usuario, únicamente necesitamos de una instancia de la clase *Candidate* donde se rellenarán todos los datos personales. Pese a ser unitarias, por convención, el controlador se definirá en plural, y la restricción se añadirá en las rutas (se explicará más adelante).

Son los siguientes:

- **CandidatesController**. Corresponiente a la clase `Candidate`
  - Acciones: *Create, New, Edit, Show, Update*
- **GreScoresController**. Correspondiente a la clase `GreScore`
  - Acciones: *Create, New, Edit, Show, Update, Destroy*
- **IeltsScoresController**. Correspondiente a la clase `IeltsScore`
  - Acciones: *Create, New, Edit, Show, Update, Destroy*
- **ToeflScoresController**. Correspondiente a la clase `ToeflScore`
  - Acciones: *Create, New, Edit, Show, Update, Destroy*

6.3.2. Clases con varias instancias

En nuestra aplicación tendremos otra serie de Clases de las que sí requerimos de varias instancias. Por ejemplo la Clase Degree, tendrá tantas instancias como títulos tenga el estudiante (Grado, Máster o cualquier otra cualificación). Su funcionamiento se define en los siguientes controladores:

- **UsersController**. Correspondiente a la clase `User`
  - Acciones: *Create, New, Edit, Show, Update, Destroy, Cancel*
- **CallsController**. Correspondiente a la clase `Call`
  - Acciones: *Index, Show, Create, Submit*
- **ApplicationsController**. Corresponiente a la clase `Application`
  - Acciones: *Index, Show, Create, Submit*
- **DegreesController**. Corresponiente a la clase `Degree`
  - Acciones: *Index, Create, New, Edit, Show, Update, Destroy, Delete*
- **PreviousPositionsController**. Correspondiente a la clase `PreviousPosition`

  - Acciones: *Index, Create, New, Edit, Show, Update, Destroy, Delete*

- **ReferencesController**. Correspondiente a la clase `Reference`
  - Acciones: *Index, Create, New, Edit, Show, Update, Destroy, Delete*
- **ResearchInterestsController**. Correspondiente a la clase `ResearchInterest`
  - Acciones: *Index, Create, New, Edit, Show, Update, Destroy, Delete*
- **CurriculumsController**. Correspondiente a la clase `Curriculum`
  - Acciones: *Index, Create, New, Edit, Show, Update, Destroy, Delete*
- **AcademicTranscriptsController**. Correspondiente a `AcademicTranscript`
  - Acciones: *Index, Create, New, Edit, Show, Update, Destroy, Delete*
- **SupervisorsController**. Correspondiente a la clase `Supervisor`
  - Acciones: *Index, Create, New, Edit, Show, Update, Destroy, Delete*

6.3.3. CRUD y resourceful Routes

Para trabajar con la base de datos debemos poder crear registros, modificarlos, eliminarlos y mostrarlos, ya sea en lista o individualmente. Con *Rails* esta tarea se puede realizar de forma sencilla y ordenada gracias a *CRUD* y *Resource Routing*.

Resource Routing permite declarar todas las rutas comunes para un controlador CRUD estándar. En lugar de declarar todas las rutas por separado, se declara una única línea que mapea las peticiones HTTP y URL en acciones del controlador. Tendremos cuatro tipos de peticiones:

- **GET** Se utiliza para mostrar información
- **POST** Para crear nuevos registros en la base de datos
- **PUT** Modifica registros en la base de datos
- **DELETE** Elimina registros

6.3.3.1. Resources y Singular Resources

Como comenté al comienzo de la sección, hay una distinción entre las clases con una instancia o con varias. De cara a la generación y mapeo de rutas, existirá una única diferencia en la declaración del resourceful routing:

```
resource     :candidates            #Para singular resources
resources    :degrees               #Para el resto
```

La diferencia entre ambas declaraciones es que, los singular resources, no necesitan referenciar un ID para mapear una ruta.

Por ejemplo, para el caso de nuestro controlador `DegreesController`, se generarán las siguientes rutas:

```
    degrees     GET     /degrees(.:format)              degrees#index
                POST    /degrees(.:format)              degrees#create
 new_degree     GET     /degrees/new(.:format)          degrees#new
edit_degree     GET     /degrees/:id/edit(.:format)     degrees#edit
     degree     GET     /degrees/:id(.:format)          degrees#show
                PUT     /degrees/:id(.:format)          degrees#update
                PATCH   /degrees/:id(.:format)          degrees#update
                DELETE  /degrees/:id(.:format)          degrees#destroy
```

Mientras que para un singular resource como Candidate, serán las siguientes:

```
    candidate     POST   /candidate(.:format)           candidates#create
new_candidate     GET    /candidate/new(.:format)       candidates#new
edit_candidate    GET    /candidate/edit(.:format)      candidates#edit
                  GET    /candidate(.:format)           candidates#show
                  PATCH  /candidate(.:format)           candidates#update
                  PUT    /candidate(.:format)           candidates#update
                  DELETE /candidate(.:format)           candidate#destroy
```

6.3.4. Acciones del controlador

Las acciones principales que tomará el controlador son las siguientes:

- **show** Muestra los atributos del objeto.
- **new** Muestra un formulario para la creación de un nuevo objeto.
- **create** Es la acción que se ejecuta como respuesta al formulario new. Crea el objeto y lo añade. No tiene una vista propia que mostrar.
- **edit** Muestra el formulario de edición de objeto.
- **update** Es la acción que se ejecuta como respuesta al formulario *edit*. Actualiza el objeto. No tiene vista propia.
- **delete** Pregunta al usuario si desea eliminar el objeto correspondiente.
- **destroy** Elimina el objeto solicitado.

No obstante, existen acciones personalizadas y que requieren tratarse de modo distinto, como la acción **submit** en el controlador *Application*. Cuando se ejecuta, se inician una serie de procesos de cara al procesamiento del formulario en la aplicación de administración.

Una vez se ejecuta, el formulario no podrá modificarse pero sí consultarse, y tanto el usuario como el personal de administración recibirán un correo informando de la aplicación.

6.3.5. Control de acceso

En *Rails*, es necesario definir qué atributos van a utilizarse en cada controlador, evitando así exponer accidentalmente atributos no deseados.

Además, dichos parámetros pueden identificarse como `required` para evitar excepciones.

Por tanto, en cada controlador marcaremos con `required` la clase a utilizar, y con `permit` los parámetros de dicha clase de los que se va a hacer uso en cada controlador.

A modo de ejemplo, para nuestro `DegreesController` tendremos lo siguiente:

```
def degree_params
    params.require(:degree).permit( :title, :institution,
    :institution_country, :subject, :grade, :degree_type,
    :year_from, :year_to )
end
```

Capítulo 7

Diseño de las vistas

7.1. Introducción

En esta sección se explicarán las *en*vistas asociadas a los *en*controladores explicados en el capítulo anterior y sus funcionalidades.

7.2. Objetivos

El objetivo de este capítulo es mostrar de forma gráfica el funcionamiento de la aplicación, siendo las *vistas* la interfaz principal con la que interactuará el usuario. Se detallarán a continuación las principales *vistas* y acciones dentro de ellas.

7.3. Vistas principales

En las *vistas* de la aplicación se muestran los datos del *modelo* gracias al *controlador*. Las *vistas* se generan utilizando código *HTML* con código *Ruby* embebido. Esto nos permite crear ficheros *HTML* dinámicos, y se guardarán bajo la extensión `.html.erb`. Tendremos dos tipos de sentencias *Ruby* principales:

- `<% @users.each do |user| %>`

  Se utilizará para ejecutar código *Ruby* sin que se muestre en el *HTML*, como por ejemplo cargar un objeto, o realizar bucles.

- `<%= user.email %>`

  Mostrará el elemento dentro de los caracteres como código *HTML*.

En el apéndice **C.3.3** se muestra cómo se desarrolla una *vista* con este tipo de código. Por simplicidad, vamos a omitir la parte del código correspondiente a las *vistas* y nos centraremos en el funcionamiento de la aplicación.

7.3.1. Sign In y Sign Up

La *gema* de autentificación *devise* se encarga del manejo de los usuarios e irá asociado al controlador `UsersController`. Lo primero que verá el usuario será la vista de *Sign In* y *Sign Up*.

En el momento de crear la cuenta o realizar la acción *Sign Up*, el usuario recibirá un e-mail con un enlace de verificación a su correo electrónico. Esta acción es obligatoria, y mientras no se realice, el usuario no podrá acceder a la aplicación.

Las acciones descritas en el *controlador* se ejecutan y presentan bajo la ruta `/users/`.

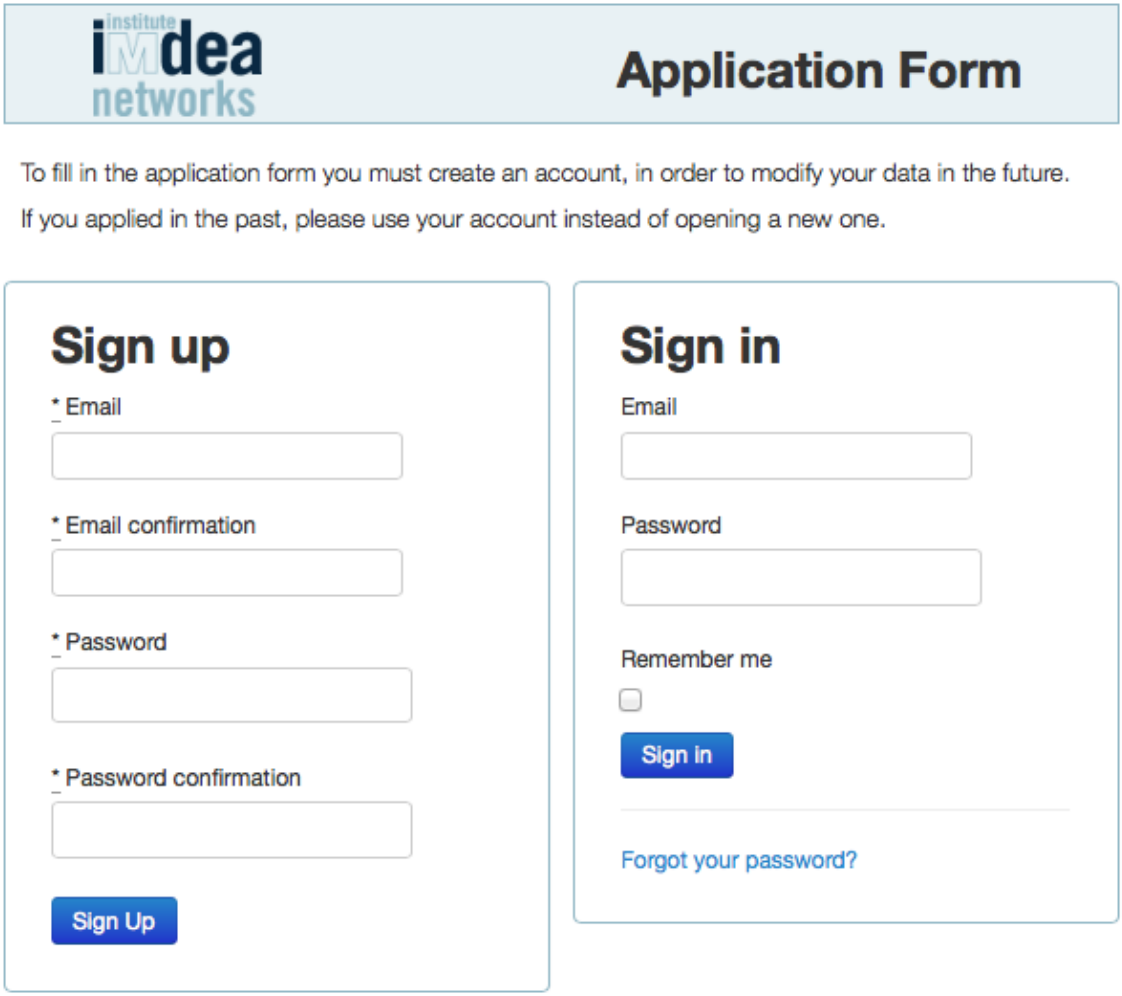


Figura 7.1: Vista de las acciones Sign in y Sign up

7.3.2. Dashboard

El *Dashboard* será la *vista* principal de la aplicación. En ella se mostrará un mensaje introductorio y las *Calls* disponibles así como su estado.

Se encuentra asociado al controlador `HomeController`, pero dada su simplicidad y que sólo dispone del método *Dashboard*, se ha omitido en el apartado anterior. La primera vez que el usuario acceda, se mostrará un mensaje para que rellene sus datos personales para poder continuar.

La única acción que toma se ejecuta sobre la ruta `/`, y será la acción `/dashboard/`.

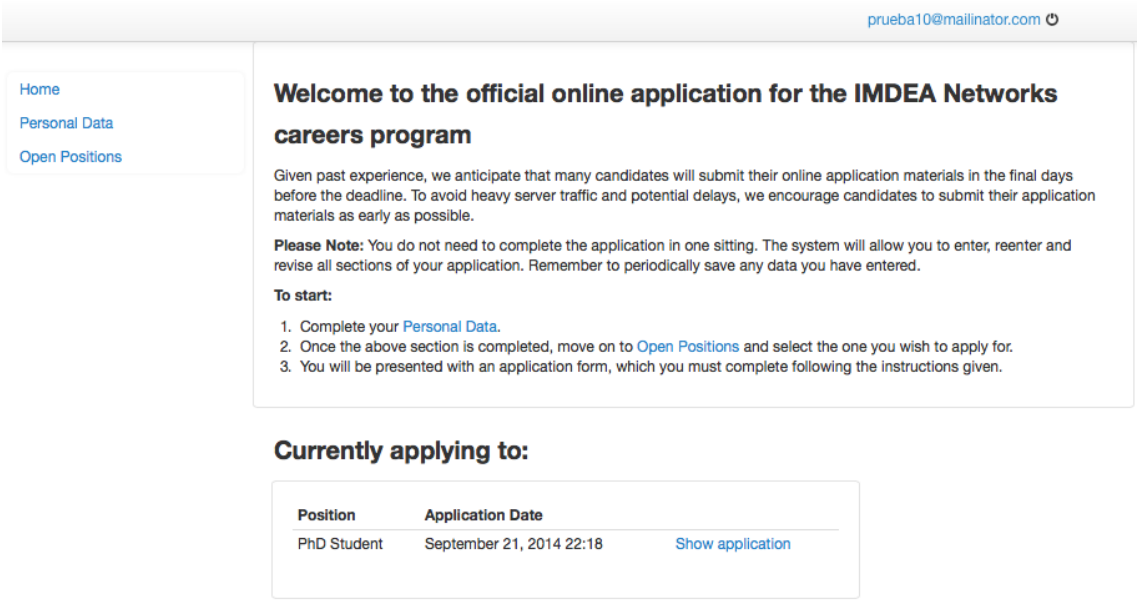


Figura 7.2: Dashboard o vista principal de la página

7.3.3. Candidate

La vista *Candidate* irá asociada al *controlador* `CandidateController`. Al existir una sóla instancia del mismo, todas las acciones se ejecutarán sobre la misma plantilla o página.

Es necesario que esta sección se complete para poder continuar. La decisión de diseño de esta forma se ha hecho pensando en que un mismo usuario puede decidir aplicar a más de una *Call*. De ese modo, y al haber separado los datos personales de la aplicación, no será necesario que lo rellene de nuevo en el caso de querer aplicar a otra posición. Se ejecuta sobre la ruta `/candidate/`, así como todas sus acciones.

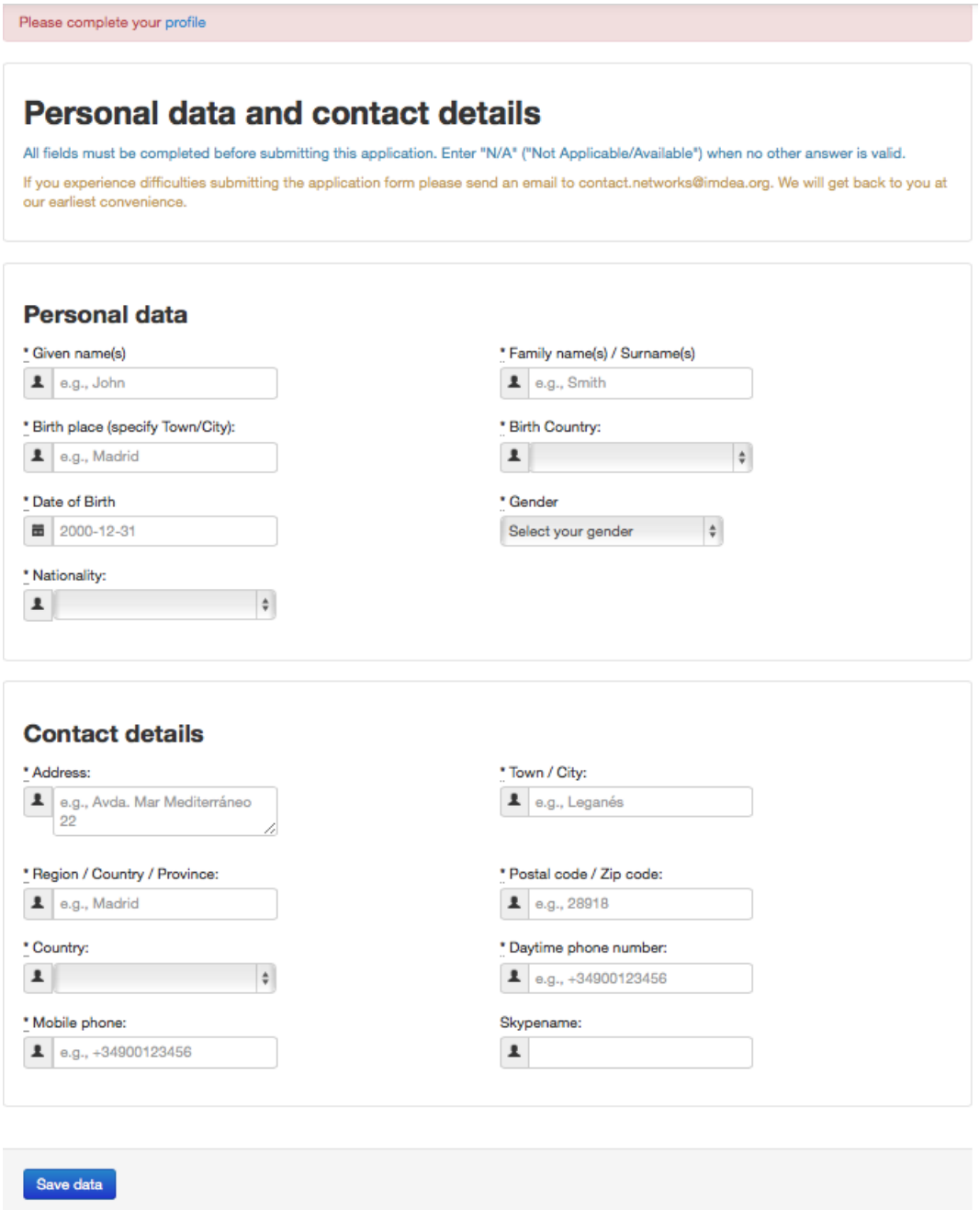


Figura 7.3: Vista Candidate que contiene los datos personales del usuario

7.3.4. Open Positions

Mostrará las posiciones que el instituto tiene abiertas a contratación. En ella y una vez haya rellenado correctamente sus datos personales (o vista *Candidate*) el usuario podrá aplicar a la posición para así acceder al formulario y rellenar sus datos.

Ejecuta la acción *index* del *controlador* `CallsController` bajo la ruta `/open_positions/`.

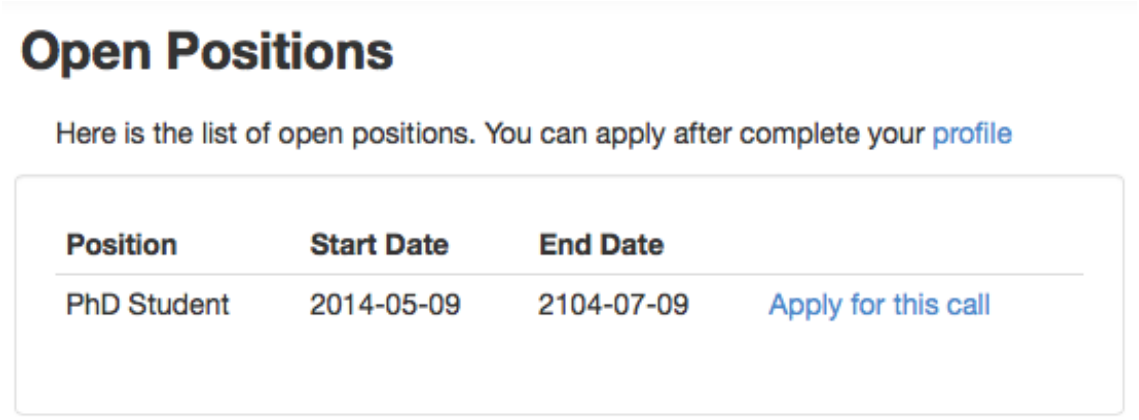


Figura 7.4: Lista de posiciones abiertas a contratación

Una vez se presione el botón de "Apply for this call", se mostrará una nueva *vista* también asociada al *controlador* `CallsController` que mostrará los detalles de la *Call* o posición. Una vez se confirme, el formulario será accesible.

Las acciones *show, create y submit* se ejecutan bajo la ruta `/apply/:id/` en el mismo controlador.

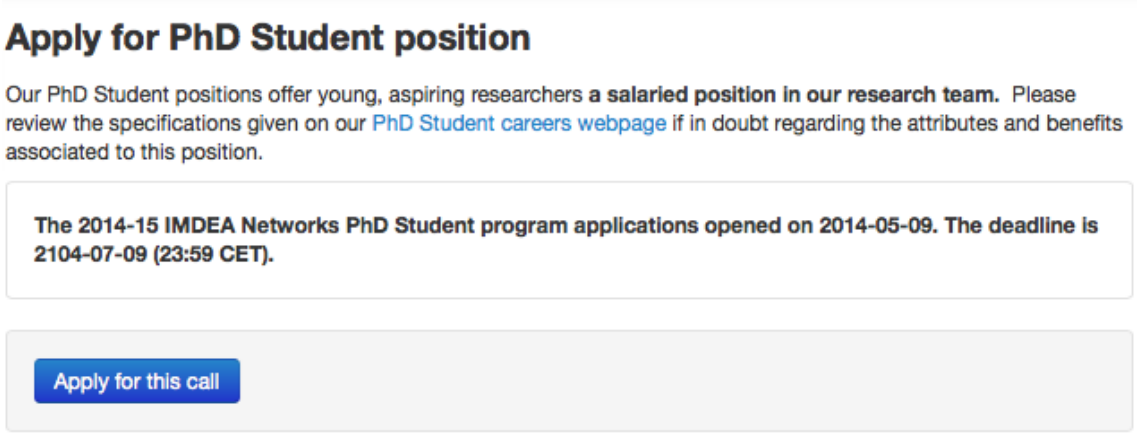


Figura 7.5: Vista de "Submit" de la aplicación a la posición

7.3.5. Application

Application mostrará todas las secciones a rellenar del formulario. Se proporcionan unos *Guidelines* y se mostrará el estado de cada parte, siendo imposible hacer efectiva la aplicación hasta que todos los datos hayan sido rellenados.

Está asociada al *controlador* `ApplicationsController`. Cada una de las partes requiere ser cumplimentada de forma distinta, por ello cada sección tiene unas instrucciones correspondientes.

En lo que al enrutamiento respecta, sus acciones se encuentran bajo rutas del tipo `/application/:application_id/`, así como el resto de *controladores* anidados a *Application*.

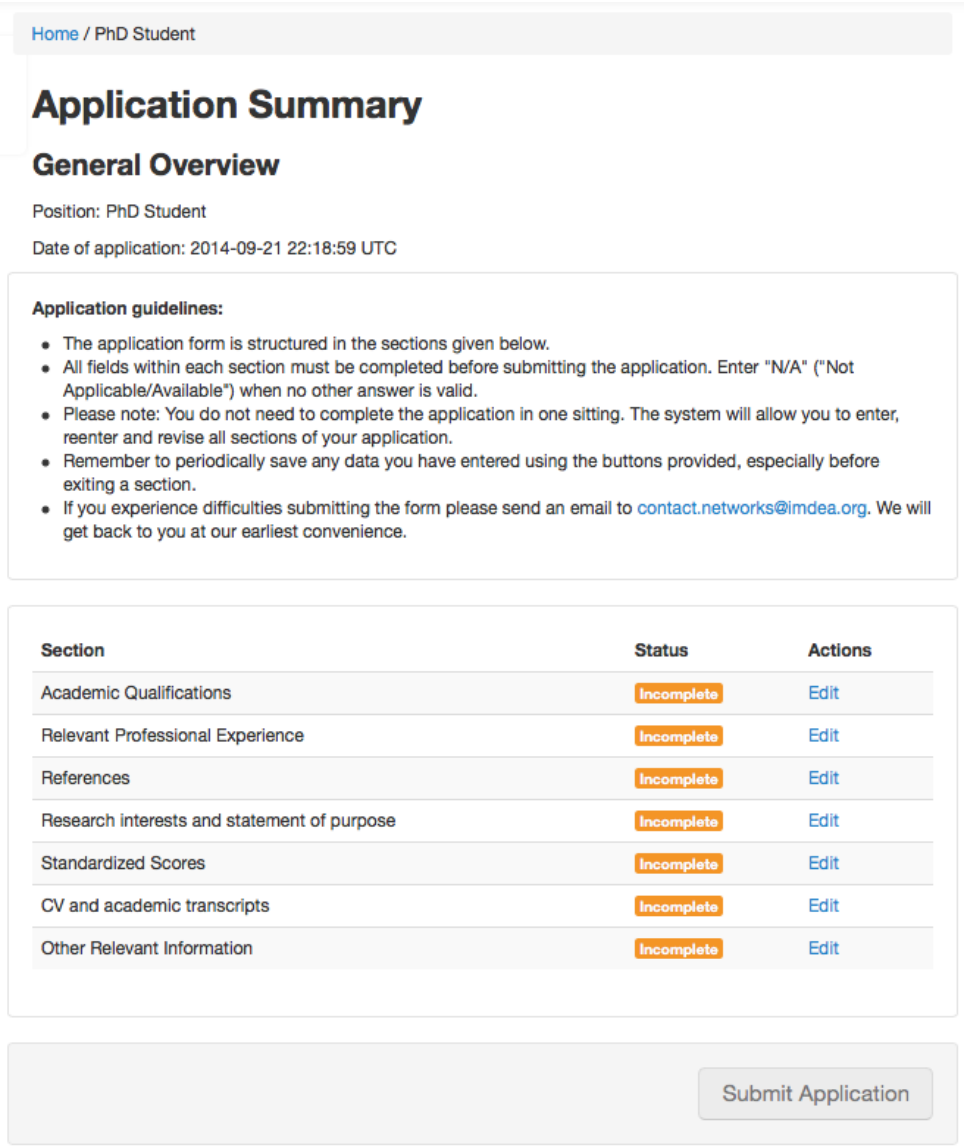


Figura 7.6: Vista principal de la aplicación

7.3.6. Academic Qualifications, Professional Experience y References

Son las tres primeras secciones del formulario, y dado que su funcionamiento es prácticamente idéntico, se resumirán en este apartado.

De ahora en adelante todas las rutas de las *en*vistas mostradas serán recursos anidados de *en*Application, y sus acciones se encuentran bajo rutas del siguiente tipo:

`/application/:application_id/degrees/` para las rutas de *Academic Qualifications*,

`/application/:application_id/previous_positions/` para las rutas de *Previous Positions*,

y `/application/:application_id/references/` para las rutas de *References*.

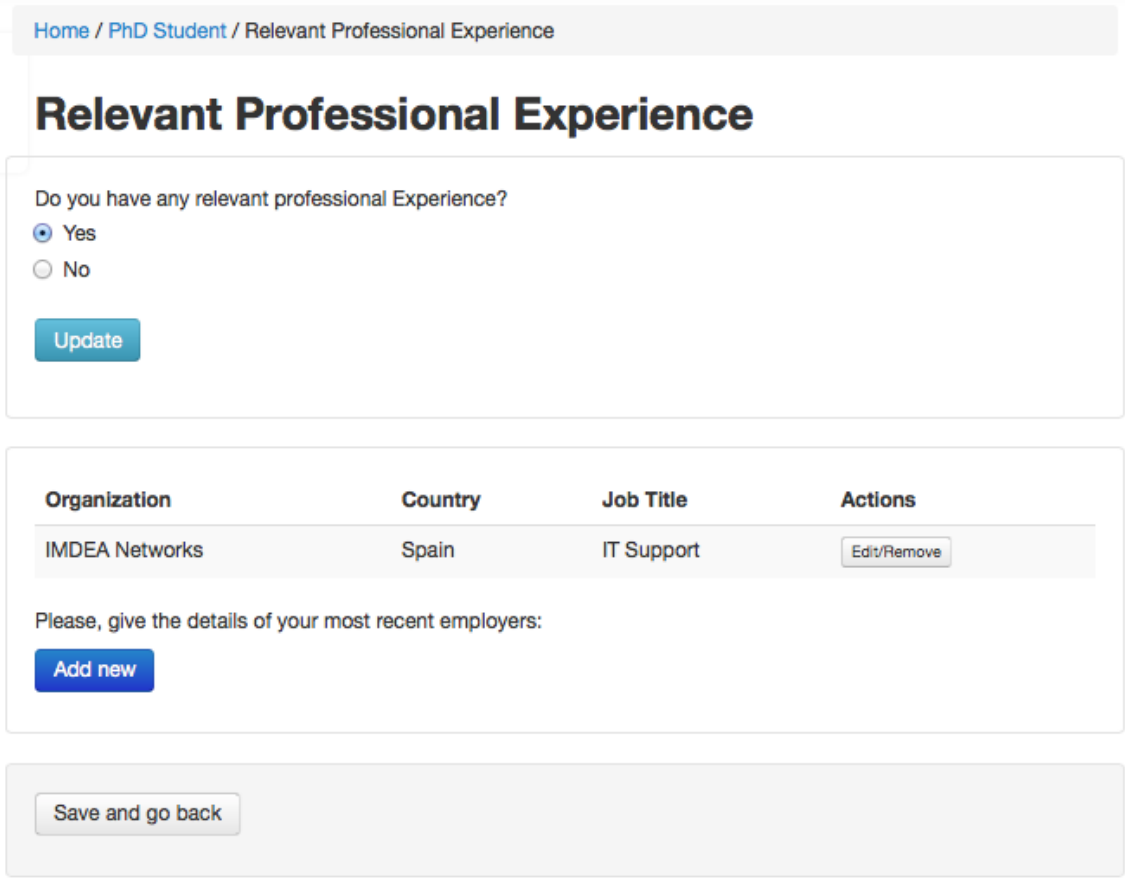


Figura 7.7: Vista Index de Relevant Profesional Experience

En determinados apartados, existe la posibilidad de que el usuario no tenga datos que cumplimentar. En ese caso, y como podemos observar en la figura anterior, se realiza una validación con un *booleano* para evitarnos crear el objeto en la base de datos y que las validaciones funcionen correctamente.

7.3.7. Research Interests and statement of purpose

En esta sección se han juntado varios *controladores* en una misma *vista*. Se compone de diferentes botones, y cada uno responderá a la acción correspondiente a su *controlador*. La única diferencia con los anteriores son las acciones que toma cada uno y el diseño, pues así se requirió en las especificaciones de la organización.

En esta *vista* podemos encontrar dos *controladores* principales que ejecutan sus acciones:

ResearchInterestsController, en `/application/:application_id/research_interests/`,

SupervisorsController, bajo la ruta `/application/:application_id/supervisors/`.

El caso de `SupervisorsController` es algo más especial. En el se podrá seleccionar un investigador del instituto y las razones de querer trabajar con él. La lista de *Supervisors* puede variar, y se modifica únicamente a través de la aplicación de administración, sincronizada con esta.

Dada la longitud de esta página, se decidió renderizar la vista relacionada con *Supervisors* con *JavaScript*, en la que para rellenar los datos aparece una ventana de tipo emergente.

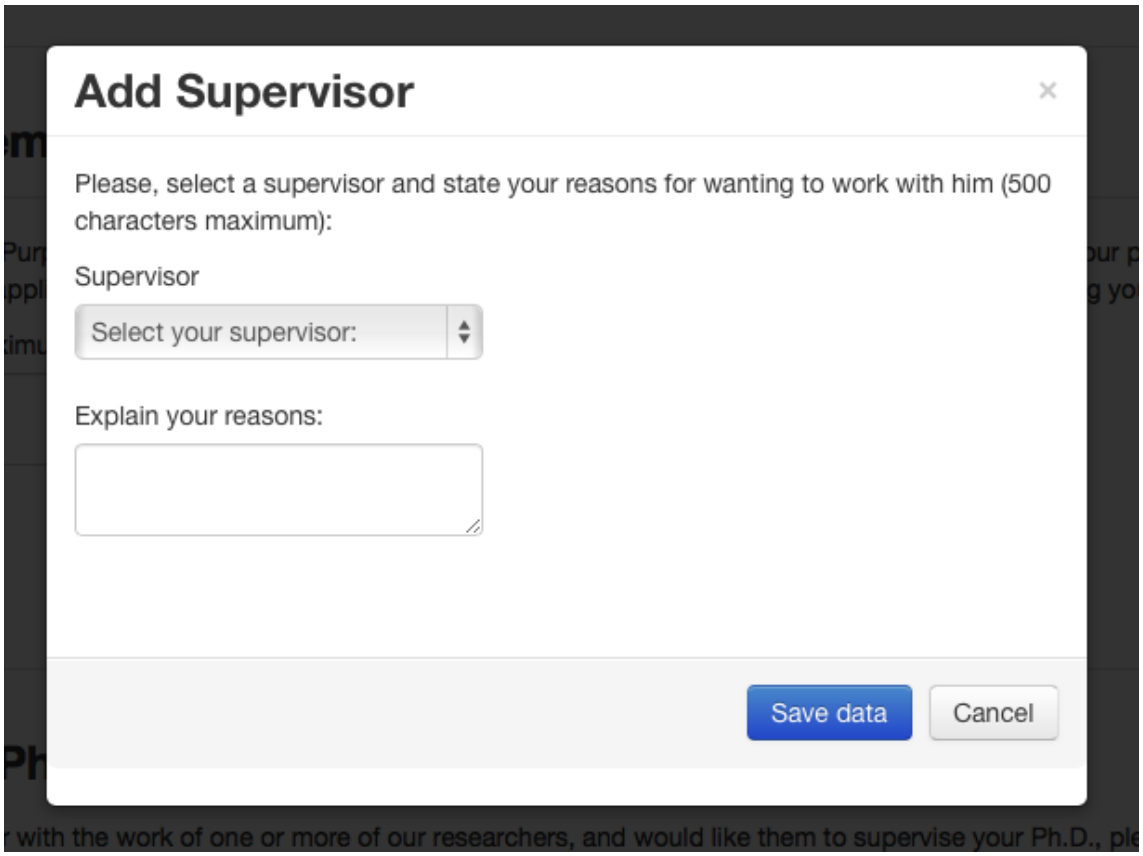


Figura 7.8: Vista Supervisors mostrada al añadir un nuevo elemento

7.3.8. Standarized Scores

De nuevo, en esta sección se agrupan tres *controladores*, los correspondientes a las calificaciones TOEFL, IELTS y GRE. El primer dato es el nivel de inglés, y en función del valor seleccionado, se renderizarán unas *vistas* u otras. Por ejemplo, si el usuario posee un nivel de inglés nativo, no será necesario que adjunte calificaciones TOEFL o IELTS.

Estas *vistas* se mostrarán de forma conjunta en la ruta `/application/:application_id/standarized_scores` y utilizando el mismo método que para renderizar *Supervisors*, mediante ventanas de tipo emergente.

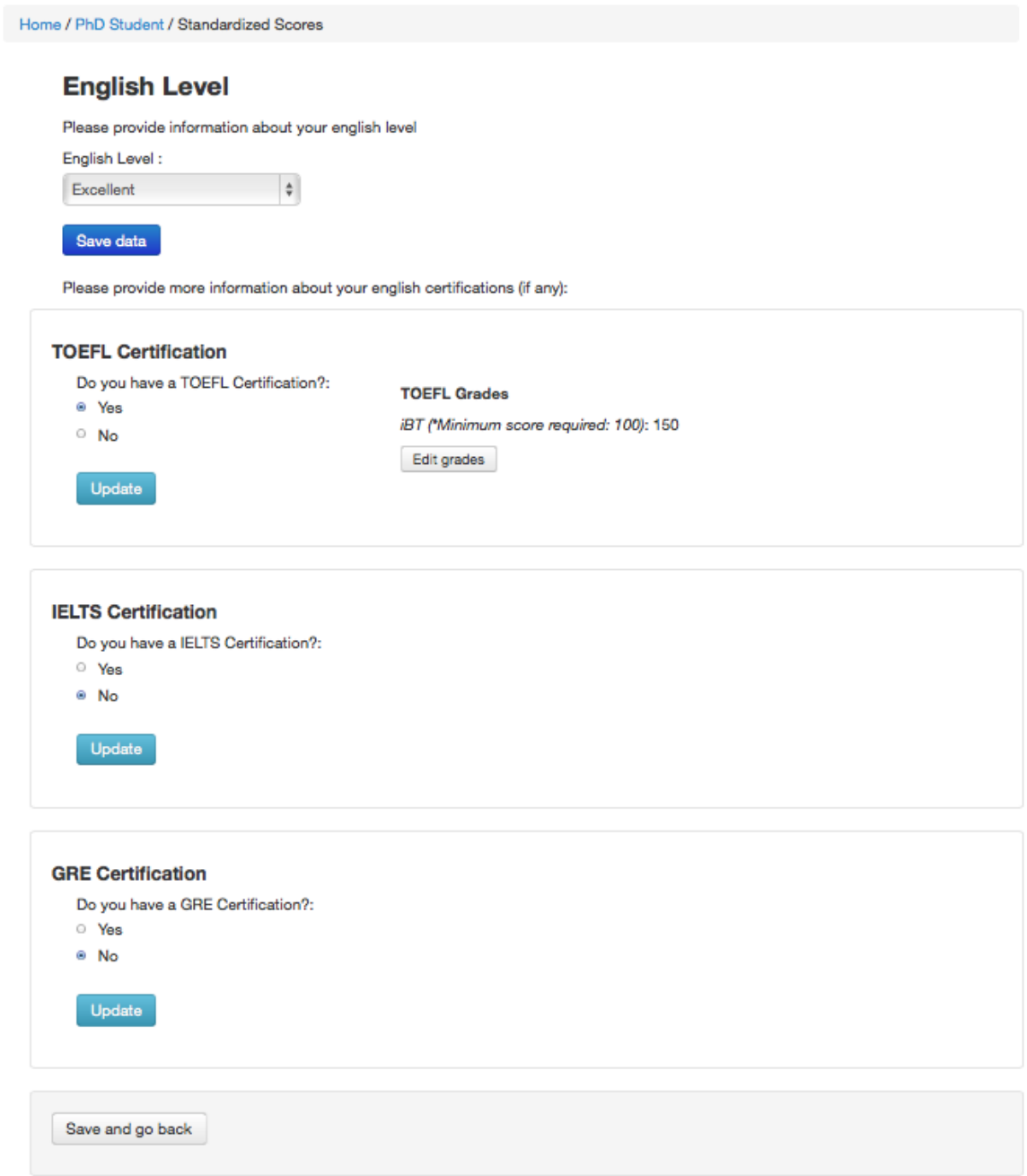


Figura 7.9: Vista Standarized Scores referente a conocimientos de la lengua inglesa

7.3.9. Documents

Se encargará de renderizar las *vistas* de los *controladores* correspondientes a la subida de documentos.

La *vista* principal mostrará el estado y los documentos que han sido añadidos, pero se distinguirá del tipo de documento, es decir, si se trata de *Currículum* o *Academic Transcripts*. Las rutas correspondientes a cada uno de los archivos son:

`/application/:application_id/documents/academic_transcripts/` para *Academic Transcripts*,

y `/application/:application_id/documents/curriculums/` para los *Currículums*.

Los archivos deben cumplir con los requisitos especificados en la *vista* (tamaño y formato), y de no ser así, la aplicación mostrará un mensaje de error.

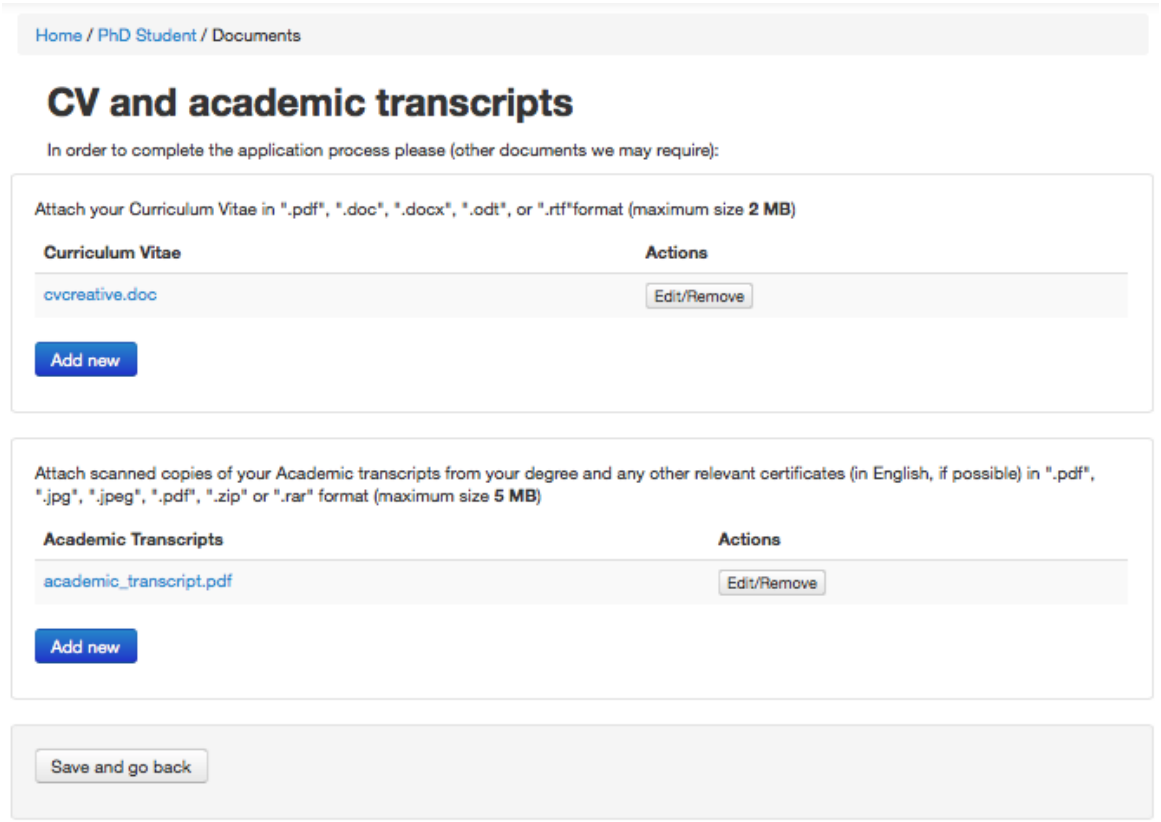


Figura 7.10: Vista Documents donde se muestran los archivos añadidos

7.3.10. Other relevant information

Es la última sección del formulario y se muestra bajo la siguiente ruta:
`/application/:application_id/other_relevant_information`

Se encarga de mostrar campos de la tabla `Other`. El funcionamiento y estética es prácticamente idéntico a secciones anteriores, por lo que en este caso vamos a omitir entrar más en detalle.

7.3.11. Acción Submit

Una vez completados todos los datos, la *vista* principal de *Application* se modificará mostrando el botón que permite hacer efectiva la entrega del formulario. Para que esto ocurra, se comprueba que todos los campos se hayan rellenado siguiendo las instrucciones provistas a lo largo del formulario. La *vista* se mostrará del siguiente modo:

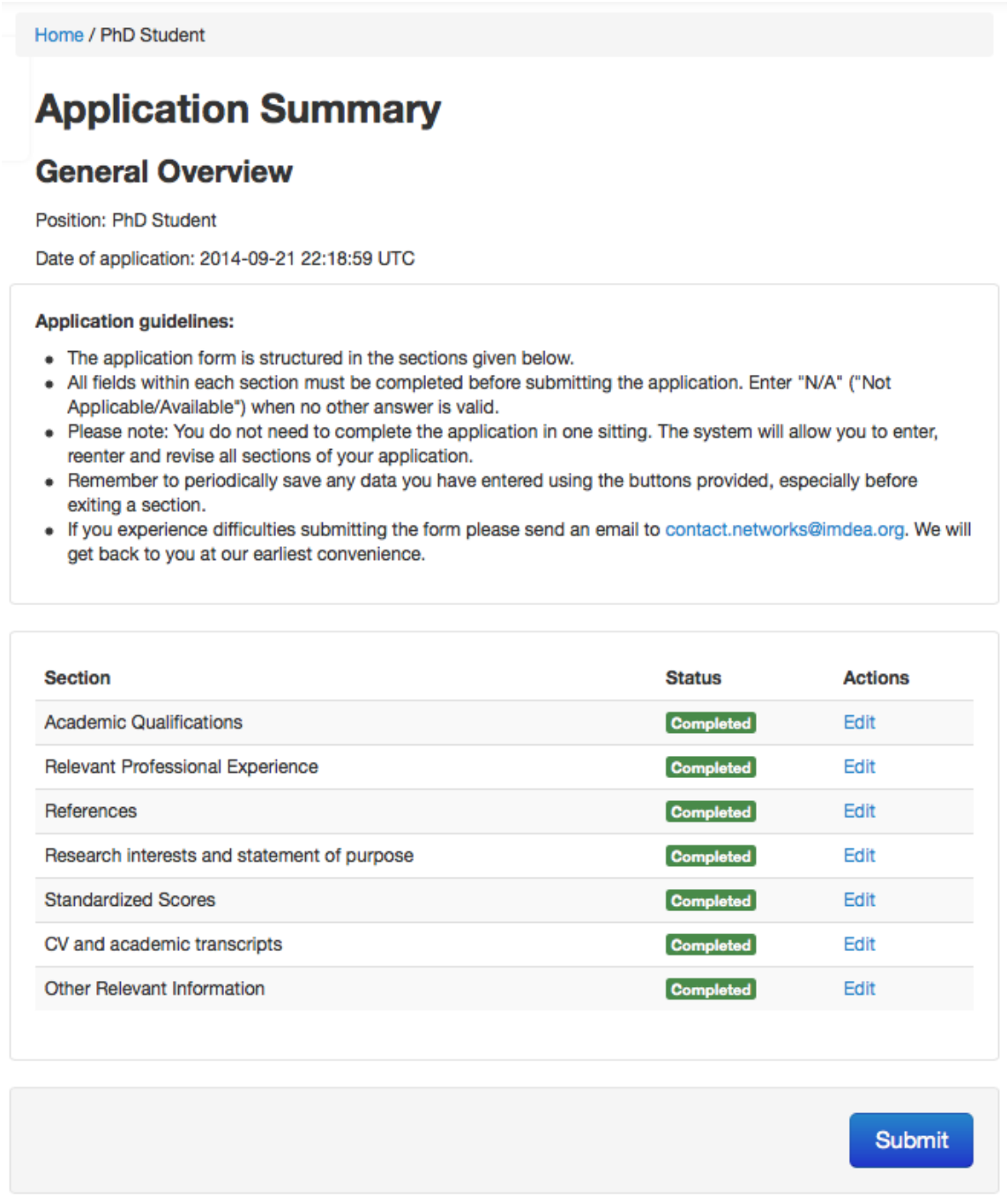


Figura 7.11: Vista Application tras completar todos los campos

Haciendo click en el botón **submit**, la aplicación nos redirigirá a otra página de confirmación donde se declara la conformidad del usuario. Si el usuario está de acuerdo y confirma la aplicación, habrá terminado el proceso.

En ese instante se renderizará de nuevo la vista *Application* modificada, donde aparecerán los campos para consultarse pero no podrán modificarse, así como un enlace para descargar un archivo PDF con todos los datos de la aplicación.

Por último, el usuario recibirá un e-mail de confirmación así como el departamento de administración confirmando que la aplicación del usuario se ha hecho efectiva.

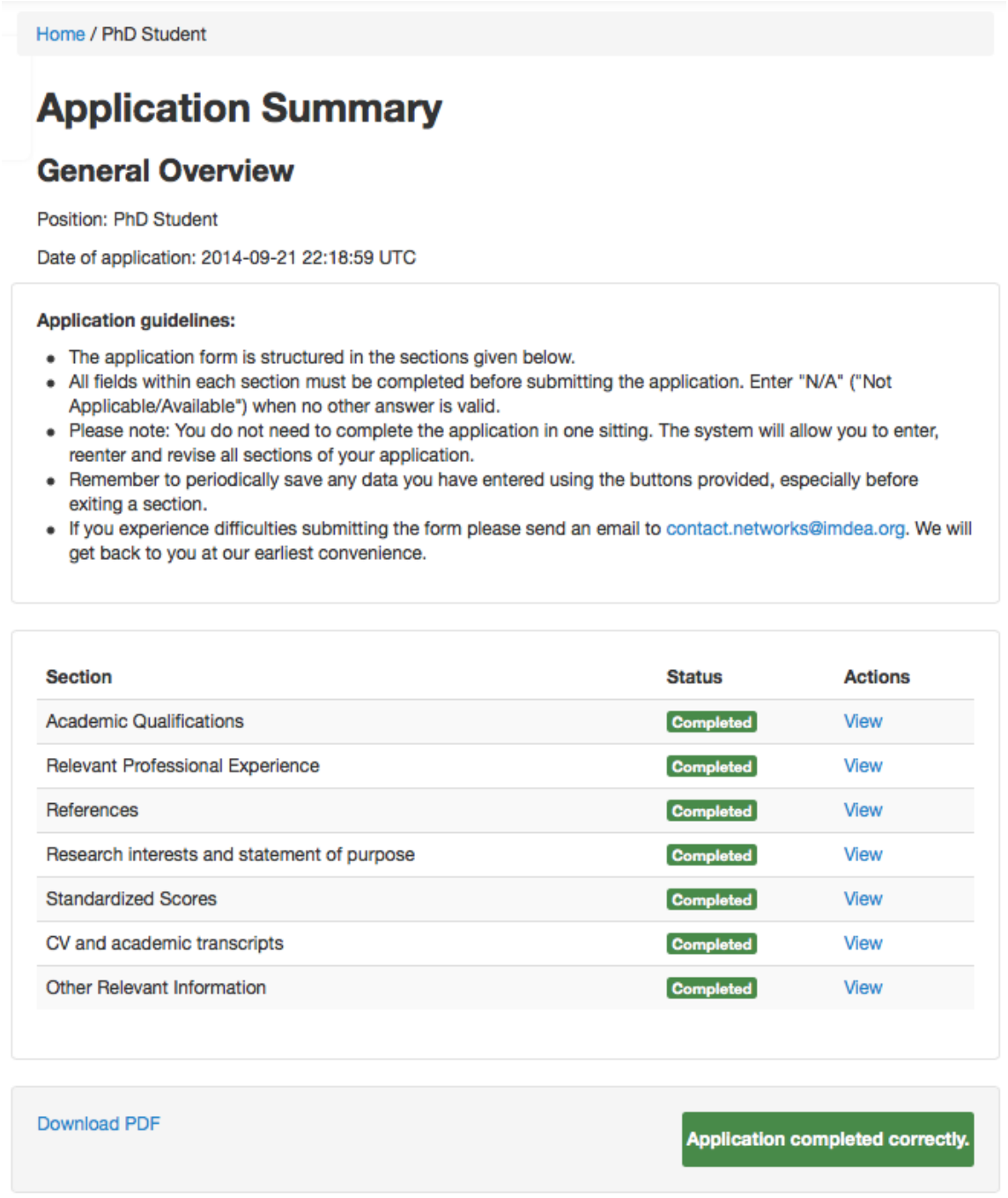


Figura 7.12: Vista Application tras haber enviado el formulario

7.4. Layouts

Durante toda la aplicación se hará uso de *layouts* o plantillas. Su uso nos permite mostrar contenido dependiendo de la parte de la aplicación en la que nos encontremos. Los principales *layouts* de los que hacemos uso son:

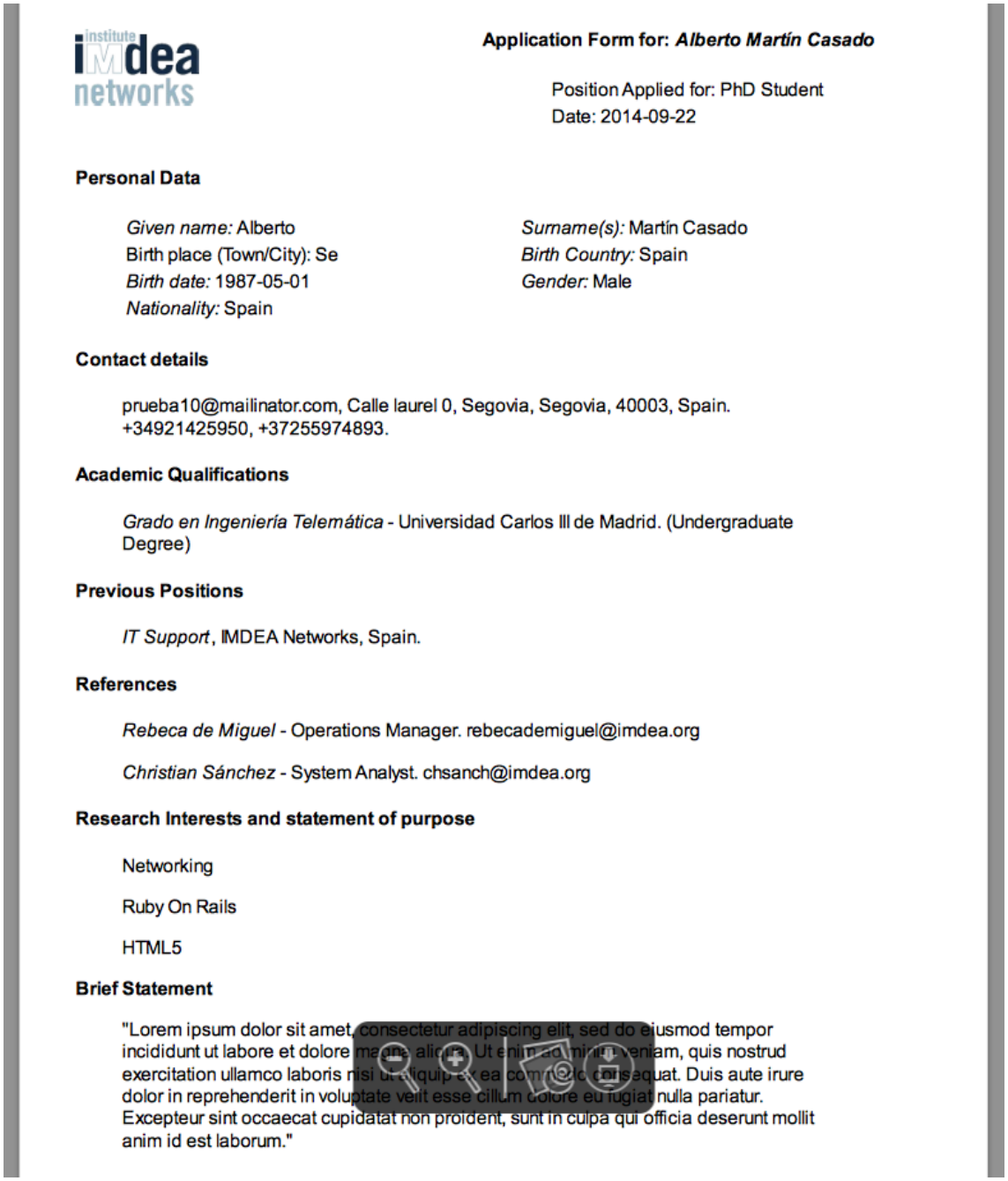
- **application.html.erb** Se trata de la plantilla principal. En ella se incluirán todos los elementos a mostrar. Algunos elementos son estáticos, como menús de navegación; y otros dinámicos dependiendo de la ruta donde nos encontremos.
- **navigation.html.erb** Muestra una barra superior donde podemos acceder a nuestros datos de usuario, así como cerrar sesión.
- **navigation_menu.html.erb** Muestra un menú lateral con las diferentes secciones de la aplicación.
- **messages.html.erb** Muestra los mensajes correspondientes a las acciones que vamos tomando.
- **pdf.html.erb** Plantilla con las cabeceras de codificación HTTP necesarias para la generación del documento PDF.

7.5. Generación de PDF

Para la generación del documento PDF hemos necesitado del uso de dos gemas, *wkhtmltopdf* y *wicked_pdf*. La primera de ellas consiste en una librería que nos permite transformar documentos HTML en PDF. La segunda, nos proporciona las herramientas necesarias para ello.

Para la generación del mismo, se ha hecho uso de un controlador `PdfGeneratorController` con una única acción, *Show*, y en el que se cargarán todos los objetos de la aplicación necesarios para mostrar los datos.

Todos esos datos son renderizados en una *vista* correspondiente con la ruta `/application/:application_id/submission_form`, y la gema *wicked_pdf* se encarga de convertirla en un documento PDF apto para la descarga del usuario, como se puede observar en la figura mostrada en la siguiente página.



institute imdea networks

Application Form for: *Alberto Martín Casado*

Position Applied for: PhD Student
Date: 2014-09-22

Personal Data

Given name: Alberto
Birth place (Town/City): Se
Birth date: 1987-05-01
Nationality: Spain

Surname(s): Martín Casado
Birth Country: Spain
Gender: Male

Contact details

prueba10@mailinator.com, Calle laurel 0, Segovia, Segovia, 40003, Spain.
+34921425950, +37255974893.

Academic Qualifications

Grado en Ingeniería Telemática - Universidad Carlos III de Madrid. (Undergraduate Degree)

Previous Positions

IT Support, IMDEA Networks, Spain.

References

Rebeca de Miguel - Operations Manager. rebecademiguel@imdea.org

Christian Sánchez - System Analyst. chsanch@imdea.org

Research Interests and statement of purpose

Networking

Ruby On Rails

HTML5

Brief Statement

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Figura 7.13: Ejemplo de documento PDF generado por la aplicación

Parte IV

Conclusiones y trabajos futuros

Capítulo 8

Conclusiones y trabajos futuros

8.1. Conclusiones

El objetivo de este Trabajo de Fin de Grado ha sido el desarrollo y puesta en funcionamiento de una aplicación web para la aplicación de estudiantes de doctorado. Este trabajo forma parte de el proyecto conocido dentro de la empresa como ARP *(Application Recruitment Process)*, que consta de dos aplicaciones Web, siendo ésta una de ellas.

El desarrollo comenzó analizando los requerimientos propuestos por la empresa para el desarrollo de la herramienta. Se estudió que herramientas podian adaptarse mejor a esos requerimientos, y finalmente se optó por elegir el *framework* de desarrollo *Ruby On Rails*.

Una vez seleccionado el *framework*, se diseñaron los casos de uso de la aplicación y se diseñó la estructura de datos basándose en la propuesta de la empresa. Sobre esa base, se comenzaron a diseñar las vistas y el comportamiento de los *controladores* de la aplicación. Una vez realizado, se integraron *modelo*, *vista* y *controlador* comprobando que el funcionamiento era correcto.

En el momento en el que tuvimos una aplicación inicial funcional, fuimos añadiendo características como automatización de tareas, notificaciones por acciones ejecutadas, correos electrónicos automatizados y procesado de documentos.

A continuación, se comenzó a probar la aplicación con datos reales. En esta parte la idea principal era dotar de robustez a la aplicación, evitando errores y comprobando que los datos eran consistentes en la base de datos. Se desarrollaron las validaciones pertinentes para ello, y se trataron aspectos de seguridad de cara a proteger los documentos que los usuarios proveen en el formulario.

Habiendo realizado las pruebas pertinentes y teniendo corregidos todos los errores mayores, se presentó a la empresa una primera versión de la aplicación. En ella, se realizó una muestra del funcionamiento de la aplicación y se recogieron todas las sugerencias y cambios propuestos.

Los cambios propuestos fueron llevados a cabo y se sincronizaron las aplicaciones, comprobando que todo funcionaba correctamente.

Con los objetivos cumplidos, la aplicación pasó del proceso de desarrollo al proceso de producción, donde se realizó la puesta a punto de la misma y su instalación en el servidor

para dotarle de acceso externo. Además, se dotó a la misma de un sistema de Back-ups, con el fin de asegurar todos los datos.

Finalmente, la aplicación final fue presentada de nuevo a la empresa, quedando conformes y dando el visto bueno para comenzar su funcionamiento.

Se concluye, por tanto, que los objetivos iniciales propuestos para este Trabajo de Fin de Grado se cumplieron.

8.2. Trabajos futuros

En la siguiente lista, se proponen varios trabajo a implementar en un futuro:

- Ampliar el formulario para otras posiciones y no sólo para puestos de doctorado. Se contempla implementarla a corto plazo. Bastaría con modificar las *vistas* y los *controladores* para, en función de la *Call*, mostrar unos datos u otros.
- Añadir *helpers* en cada uno de los apartados, dotando de explicaciones concretas en cada apartado de la aplicación.
- Mejorar la interfaz. Para el desarrollo de la aplicación se ha utilizado la versión de CSS *Bootstrap* 2.3.2. Actualmente, *Bootstrap* se encuentra en su versión 3.2.0, pero debido a la gran cantidad de cambios que requería la actualización, se pospuso para un futuro.
- Desarrollar mejoras usando código *JavaScript* en el formulario. Actualmente, por cada *controlador* se ejecuta una acción, y para que los datos tengan efecto hay que hacer click en cada sección. Haciendo uso de *JavaScript*, podría implementarse una funcionalidad de autoguardado que hiciese más sencillo completar la aplicación.

Bibliografía

[1] About python. https://www.python.org/about/.

[2] Documentación readme de la gema sqlite3. http://rubydoc.info/gems/sqlite3/1.3.9/frames.

[3] The java language environment. http://www.oracle.com/technetwork/java/intro-141325.html.

[4] Licencia gpl de ruby. https://www.ruby-lang.org/en/about/license.txt.

[5] The mit license (mit). http://opensource.org/licenses/mit-license.php.

[6] Página oficial del gestor gitlab. https://about.gitlab.com.

[7] Página oficial del paquete homebrew. http://brew.sh.

[8] Página web oficial de la librería wkhtmltopdf. http://wkhtmltopdf.org/index.html.

[9] Repositorio y documentación de la gema bootstrap-sass. https://github.com/twbs/bootstrap-sass.

[10] Repositorio y documentación de la gema breadcrumbs on rails. https://github.com/weppos/breadcrumbs_on_rails.

[11] Repositorio y documentación de la gema country-select. https://github.com/jamesds/country-select.

[12] Repositorio y documentación de la gema devise. https://github.com/plataformatec/devise.

[13] Repositorio y documentación de la gema jbuilder. https://github.com/rails/jbuilder.

[14] Repositorio y documentación de la gema jquery-fileupload-rails. https://github.com/tors/jquery-fileupload-rails.

[15] Repositorio y documentación de la gema jquery-rails. https://github.com/rails/jquery-rails.

[16] Repositorio y documentación de la gema paperclip. https://github.com/thoughtbot/paperclip.

[17] Repositorio y documentación de la gema rails. https://github.com/rails/rails.

[18] Repositorio y documentación de la gema rolify. https://github.com/RolifyCommunity/rolify.

[19] Repositorio y documentación de la gema simple form. https://github.com/plataformatec/simple_form.

[20] Repositorio y documentación de la gema uglifier. https://github.com/lautis/uglifier.

[21] Repositorio y documentación de la gema wicked pdf. https://github.com/mileszs/wicked_pdf.

[22] Repositorio y documentación de la librería rbenv. https://github.com/sstephenson/rbenv.

[23] Sitio web oficial de git. http://git-scm.com.

[24] Sitio web oficial del editor sublime text 2. http://www.sublimetext.com/2.

[25] Sitio web oficial del framework ruby on rails. http://rubyonrails.org.

[26] Sitio web oficial del lenguaje ruby. https://www.ruby-lang.org/es/.

[27] K. Beck et al. Manifesto for agile software development. http://www.agilemanifesto.org.

[28] Apple Inc. Especificaciones del sistema operativo os x. https://www.apple.com/es/osx/specs/.

[29] Apple Inc. Paquete de desarrollo xcode. https://developer.apple.com/xcode/.

[30] W. Royce. *Managing the Development of Large Software Systems*, pages 328–339. IEEE CS Press, 1970.

[31] S. Ruby. *Agile Web Development with Rails 4*, volume P2.0, page 31. The Pragmatic Bookshelf, 2013.

[32] Inc. Standish Group International. Chaos chronicles. http://www1.standishgroup.com//sample_research/chaos_1994_1.php, 1994.

[33] A. Summer. *Everyday Rails Testing with RSpec: A practical approach to test-driven development*. Leanpub, 2014.

[34] J. Sutherland. Perfil profesional, donde se incluye una lista de clientes conocidos. https://www.scrumalliance.org/community/profile/jsutherland.

Parte V

Anexos

Apéndice A

Planificación y presupuesto

En esta sección se va a presentar la descomposición de tareas que se han llevado a cabo para realizar este Trabajo Fin de Grado. Se describen cada una de las tareas así como la relación con otras tareas y el esfuerzo de cada una.

A.1. Descomposición en tareas

- **Tarea A: Análisis**
  - **Tarea A.1: Requerimientos**
    - **Descripción:** se estudió la propuesta de la organización para el desarrollo de la aplicación solicitada.
    - **Objetivos:** comprensión de la propuesta
    - **Dependencia:** da comienzo a este Trabajo de Fin de Grado.
    - **Duración:** 1 semana.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes; Ingeniero Senior 0.25 ingenieros/mes.
  - **Tarea A.2: Herramientas y framework**
    - **Descripción:** se estudiaron las herramientas a utilizar para el desarrollo.
    - **Objetivos:** estudio de las herramientas que más se ajustasen a los requerimientos y necesidades del proyecto.
    - **Dependencia:** da comienzo tras la tarea A.1.
    - **Duración:** 1 semana.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes; Ingeniero Senior 0.25 ingenieros/mes.
  - **Tarea A.3: Casos de uso**
    - **Descripción:** se diseñaron de los diferentes casos de uso para el funcionamiento de la aplicación.
    - **Objetivos:** definir la lógica de la aplicación.
    - **Dependencia:** da comienzo tras la tarea A.2.
    - **Duración:** 2,4 semanas.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes.

- **Tarea A.4: Estructura de datos**
  - **Descripción:** se diseñó la estructura de la base de datos de la aplicación.
  - **Objetivos:** tener una base de datos optimizada para los datos que la organización solicita.
  - **Dependencia:** da comienzo tras la tarea A.3.
  - **Duración:** 1,2 semanas.
  - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes; Ingeniero Senior 0.25 ingenieros/mes.

- **Tarea B: Diseño**
  - **Tarea B.1: Vistas HTML (formulario, dashboard)**
    - **Descripción:** se realizó el diseño de las vistas del formulario siguiendo las especificaciones marcadas por la organización.
    - **Objetivos:** crear una interfaz limpia e intuitiva, acorde con las necesidades.
    - **Dependencia:** da comienzo tras la tarea A.4.
    - **Duración:** 3,8 semanas.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes
  - **Tarea B.2: Diseño del modelo (Database)**
    - **Descripción:** se implementó la estructura de datos diseñada en la tarea A.4.
    - **Objetivos:** establecer las relaciones entre tablas de la base de datos así como las validaciones del lado del servidor.
    - **Dependencia:** da comienzo tras la tarea B.1.
    - **Duración:** 2 semanas.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes
  - **Tarea B.3: Diseño del controlador (funciones y casos de uso)**
    - **Descripción:** se implementaron las funcionalidades diseñadas en la tarea A.3. y las interacciones con el modelo.
    - **Objetivos:** definir todos los métodos que dotan de funcionalidad a la aplicación.
    - **Dependencia:** da comienzo tras la tarea B.2.
    - **Duración:** 4 semanas.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes

- **Tarea C: Implementación**
  - **Tarea C.1: Integración de la base de datos con las vistas y los controladores**
    - **Descripción:** se comprobó que modelo, vista y controlador funcionaban de forma síncrona y correcta.
    - **Objetivos:** tener una aplicación inicial funcional en la que se puedan almacenar datos y comprobar que las vistas actúan como se ha definido en el controlador
    - **Dependencia:** da comienzo tras la tarea B.3.
    - **Duración:** 1 semana.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes

- **Tarea C.2: Integración del sistema de autenticación**
  - **Descripción:** se integró el sistema de autenticación del instituto (LDAP) en la herramienta
  - **Objetivos:** integrar autenticación interna de la organización para acceso a las aplicaciones
  - **Dependencia:** da comienzo tras la tarea C.1.
  - **Duración:** 0,4 semanas.
  - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes; Ingeniero Senior 0.25 ingenieros/mes.
- **Tarea C.3: Automatización de tareas**
  - **Descripción:** se definieron acciones automáticas dentro de la aplicación
  - **Objetivos:** cambiar el estado de la aplicación en función de los parámetros introducidos por el aplicante, para su posterior revisión.
  - **Dependencia:** da comienzo tras la tarea C.2.
  - **Duración:** 1 semana.
  - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes
- **Tarea C.4: Notificaciones**
  - **Descripción:** se dotó a la aplicación de un sistema de alertas visuales para el usuario.
  - **Objetivos:** informar al usuario del estado de la acción tomada; si esta se ha realizado correctamente o si ha habido algún error.
  - **Dependencia:** da comienzo tras la tarea C.3.
  - **Duración:** 1 semana.
  - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes
- **Tarea C.5: Procesado de documentos**
  - **Descripción:** se instaló y adaptó una gema que permite la subida de documentos a la aplicación.
  - **Objetivos:** permitir que documentos tales como el currículum o datos académicos acompañen a la solicitud.
  - **Dependencia:** da comienzo tras la tarea C.4.
  - **Duración:** 1,6 semanas.
  - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes
- **Tarea C.6: Mailing automático**
  - **Descripción:** se configuró la aplicación para que sea capaz de enviar correos electrónicos automáticos en determinadas circunstancias.
  - **Objetivos:** verificar usuarios, notificar cuando se ha enviado la solicitud, notificar a la persona encargada en la organización acerca de dicha solicitud.
  - **Dependencia:** da comienzo tras la tarea C.5.
  - **Duración:** 1 semana.
  - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes

- **Tarea D: Testing**
  - **Tarea D.1: Carga de datos**
    - **Descripción:** se rellenaron varios formularios siguiendo procesos distintos para comprobar la funcionalidad de la aplicación

- **Objetivos:** detección de errores
- **Dependencia:** da comienzo tras la tarea C.6.
- **Duración:** 1 semana.
- **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes.

- **Tarea D.2: Validaciones**
  - **Descripción:** se implementaron las validaciones oportunas en el formulario
  - **Objetivos:** evitar recibir aplicaciones incompletas, con datos incorrectos y evitar la subida de ficheros con código malicioso
  - **Dependencia:** da comienzo tras la tarea D.1.
  - **Duración:** 1 semana.
  - **Recursos:** Ingeniero 0.25 ingenieros/mes
- **Tarea D.3: Seguridad**
  - **Descripción:** se estableció un sistema de autenficación distinto para cada usuario (login requerido), cifrado de archivos en el servidor y prevención de inyecciones SQL
  - **Objetivos:** dotar de seguridad a la aplicación, evitar recibir solicitudes automáticas o spam y proteger los datos de los usuarios.
  - **Dependencia:** da comienzo tras la tarea D.2.
  - **Duración:** 2 semanas.
  - **Recursos:** Ingeniero 0.25 ingenieros/mes
- **Tarea D.4: Corrección de errores**
  - **Descripción:** se corrigieron numerosos bugs, funcionalidades y comportamiento de vistas así como pequeñas modificaciones en la base de datos o modelo.
  - **Objetivos:** conseguir una versión estable y funcional de la aplicación.
  - **Dependencia:** da comienzo tras la tarea D.3.
  - **Duración:** 2 semanas.
  - **Recursos:** Ingeniero 0.25 ingenieros/mes

- **Tarea E: Presentación de la primera beta**
  - **Tarea E.1: Implementación de cambios**
    - **Descripción:** tras la presentación de la primera beta la organización sugirió realizar cambios en el diseño y comportamiento de la aplicación.
    - **Objetivos:** adaptar la aplicación a las necesidades de la organización.
    - **Dependencia:** da comienzo tras la tarea D.4.
    - **Duración:** 4 semanas.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes; Ingeniero Senior 0.25 ingenieros/mes.
  - **Tarea E.2: Pruebas de los cambios**
    - **Descripción:** tras la implementación de los cambios, se realizaron de nuevo pruebas para verificar el correcto funcionamiento de los cambios realizados.
    - **Objetivos:** mantener la funcionalidad y estabilidad una vez realizados los cambios
    - **Dependencia:** da comienzo tras la tarea E.1.
    - **Duración:** 3 semanas.

- **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes

- **Tarea F: Sincronización entre aplicaciones**
  - **Tarea F.1: Sincronización y carga de datos en la aplicación de administración**
    - **Descripción:** se rellenaron diferentes formularios de prueba
    - **Objetivos:** verificar el funcionamiento de la herramienta administrativa haciendo uso de los datos rellenados con anterioridad.
    - **Dependencia:** da comienzo tras la tarea E.2.
    - **Duración:** 1 semana.
    - **Recursos:** G. Ingeniería Telemática 0.5 ingenieros/mes; Ingeniero Senior 0.5 ingenieros/mes.
  - **Tarea F.2: Sincronización de documentos en la aplicación de administración**
    - **Descripción:** se adjuntaron diferentes documentos de prueba en el formulario
    - **Objetivos:** sincronizar dichos documentos con la herramienta administrativa y se comprobó la seguridad y accesibilidad a los mismos
    - **Dependencia:** da comienzo tras la tarea F.1.
    - **Duración:** 1 semana.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes; Ingeniero Senior 0.25 ingenieros/mes.
  - **Tarea F.3: Corrección de errores**
    - **Descripción:** se corrigieron errores y se reconfiguraron parámetros de cara a la sincronización de datos y documentos en la herramienta administrativa
    - **Objetivos:** dotar de una óptima sincronización a ambas aplicaciones para su correcto funcionamiento
    - **Dependencia:** da comienzo tras la tarea F.2.
    - **Duración:** 2 semanas.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes; Ingeniero Senior 0.25 ingenieros/mes.

- **Tarea G: Desarrollo y despliegue**
  - **Tarea G.1: Acceso público**
    - **Descripción:** se migró la aplicación de entorno de desarrollo a entorno de producción
    - **Objetivos:** poner la herramienta en funcionamiento
    - **Dependencia:** da comienzo tras la tarea F.3.
    - **Duración:** 0,6 semanas.
    - **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes; Ingeniero Senior 0.25 ingenieros/mes.
  - **Tarea G.2: Sistema de backups**
    - **Descripción:** se dotó a la aplicación de un sistema de back-ups
    - **Objetivos:** garantizar la persistencia de los datos en caso de errores
    - **Dependencia:** da comienzo tras la tarea G.1.

- **Duración:** 0,6 semanas.
- **Recursos:** G. Ingeniería Telemática 0.25 ingenieros/mes; Ingeniero Senior 0.25 ingenieros/mes.

- **Tarea H: Memoria.**
  - **Tarea H.1: Redacción de la memoria.**
    - **Descripción:** se redactó esta memoria.
    - **Objetivos:** obtener la memoria del Trabajo de Fin de Grado.
    - **Dependencia:** da comienzo tras la tarea G.2.
    - **Duración:** 33 días.
    - **Recursos:** G. Ingeniería Telemática 0.75 ingenieros/mes

Tarea	Duración (semanas)	G. Ing. Tlm. (Ing/m)	Ing. Senior (Ing/m)
Análisis			
A.1 Requerimientos	1	0.25	0.25
A.2 Herramientas y framework	1	0.25	0.25
A.3 Casos de uso	2.4	0.6	-
A.4 Estructura de datos	1.2	0.3	0.3
Total		**1.4**	**0.8**
Diseño			
B.1 Vistas HTML (formulario, dashboard)	3.8	0.95	-
B.2 Diseño del modelo (Database)	2	0.5	-
B.3 Diseño del controlador (funciones y casos...	4	1	-
Total		**2.45**	-
Implementación			
C.1 Integración de la base de datos con las...	1	0.25	-
C.2 Integración del sistema de autenticación	0.4	0.1	0.1
C.3 Automatización de tareas	1	0.25	-
C.4 Notificaciones	1	0.25	-
C.5 Procesado de documentos	1.6	0.4	-
C.6 Mailing automático	1	0.25	-
Total		**1.5**	**0.1**
Testing			
D.1 Carga de datos	1	0.25	-
D.2 Validaciones	1	0.25	-
D.3 Seguridad	2	0.5	-
D.4 Corrección de errores	2	0.5	-
Total		**1.5**	-
Presentación primera beta			
E.1 Implementación de cambios	4	1	1
E.2 Pruebas de los cambios	3	0.75	-
Total		**1.75**	**1**
Sincronización entre aplicaciones			
F.1 Sincronización y carga de datos en la...	1	0.25	0.25
F.2 Sincronización de documentos en la...	1	0.25	0.25
F.3 Corrección de errores	2	0.5	0.5
Total		**1**	**1**
Desarrollo			
G.1 Acceso público	0.6	0.15	0.15
G.2 Sistema de back-ups	0.6	0.15	0.15
Total		**0.3**	**0.3**
Memoria			
H.1 Desarrollo de la memoria	6.6	1.65	-
Total		**1.65**	-
Total	45.2	**11.55**	**3.2**

Tabla A.1: Resumen descomposición en tareas

A.2. Planificación con el diagrama de fases de ejecución detallado

En la figura A.1 se muestra el diagrama de Gantt con las tareas principales. Además en la figura A.2 se presenta el diagrama de Gantt con todas las tareas que se han realizado en este Trabajo Fin de Grado.

	Name	Duration
1	⊞Tarea A: Análisis	29 days
6	⊞Tarea B: Diseño	50 days
10	⊞Tarea C: Implementación	30 days
17	⊞Tarea D: Testing	30 days
22	⊞Tarea E: Presentación primera beta	35 days
25	⊞Tarea F: Sincronización entre aplicaciones	20 days
29	⊞Tarea G: Desarrollo	6 days
32	⊞Tarea H: Memoria	33 days

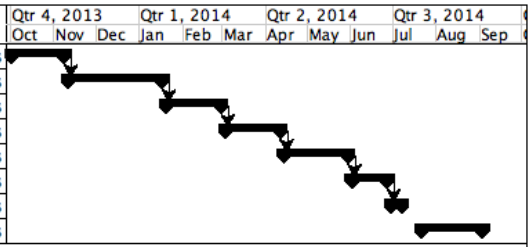


Figura A.1: Diagrama de Gantt con la planificación del proyecto resumida

	Name	Duration
1	**Tarea A: Análisis**	**29 days**
2	Tarea A1: Requerimientos	5 days
3	Tarea A2: Herramientas y framework	5 days
4	Tarea A3: Casos de uso	12 days
5	Tarea A4: Estructura de datos	6 days
6	**Tarea B: Diseño**	**50 days**
7	Tarea B1: Vistas HTML (formulario, dashboard)	19 days
8	Tarea B2: Diseño del modelo (Database)	10 days
9	Tarea B3: Diseño del controlador (funciones y casos de uso)	20 days
10	**Tarea C: Implementación**	**30 days**
11	Tarea C1: Integración de la base de datos con las vistas y los...	5 days
12	Tarea C2: Integración del sistema de autenticación	2 days
13	Tarea C3: Automatización de tareas	5 days
14	Tarea C4: Notificaciones	5 days
15	Tarea C5: Procesado de documentos	8 days
16	Tarea C6: Mailing automático	5 days
17	**Tarea D: Testing**	**30 days**
18	Tarea D1: Carga de datos	5 days
19	Tarea D2: Validaciones	5 days
20	Tarea D3: Seguridad	10 days
21	Tarea D4: Corrección de errores	10 days
22	**Tarea E: Presentación primera beta**	**35 days**
23	Tarea E1: Implementación de cambios	20 days
24	Tarea E2: Pruebas de los cambios	15 days
25	**Tarea F: Sincronización entre aplicaciones**	**20 days**
26	Tarea F1: Sincronización y carga de datos en la aplicación de...	5 days
27	Tarea F2: Sincronización de documentos en la aplicación de a...	5 days
28	Tarea F3: Corrección de errores	10 days
29	**Tarea G: Desarrollo**	**6 days**
30	Tarea G1: Acceso público	3 days
31	Tarea G2: Sistema de back-ups	3 days
32	**Tarea H: Memoria**	**33 days**
33	Tarea H1: Desarrollo de la memoria	33 days

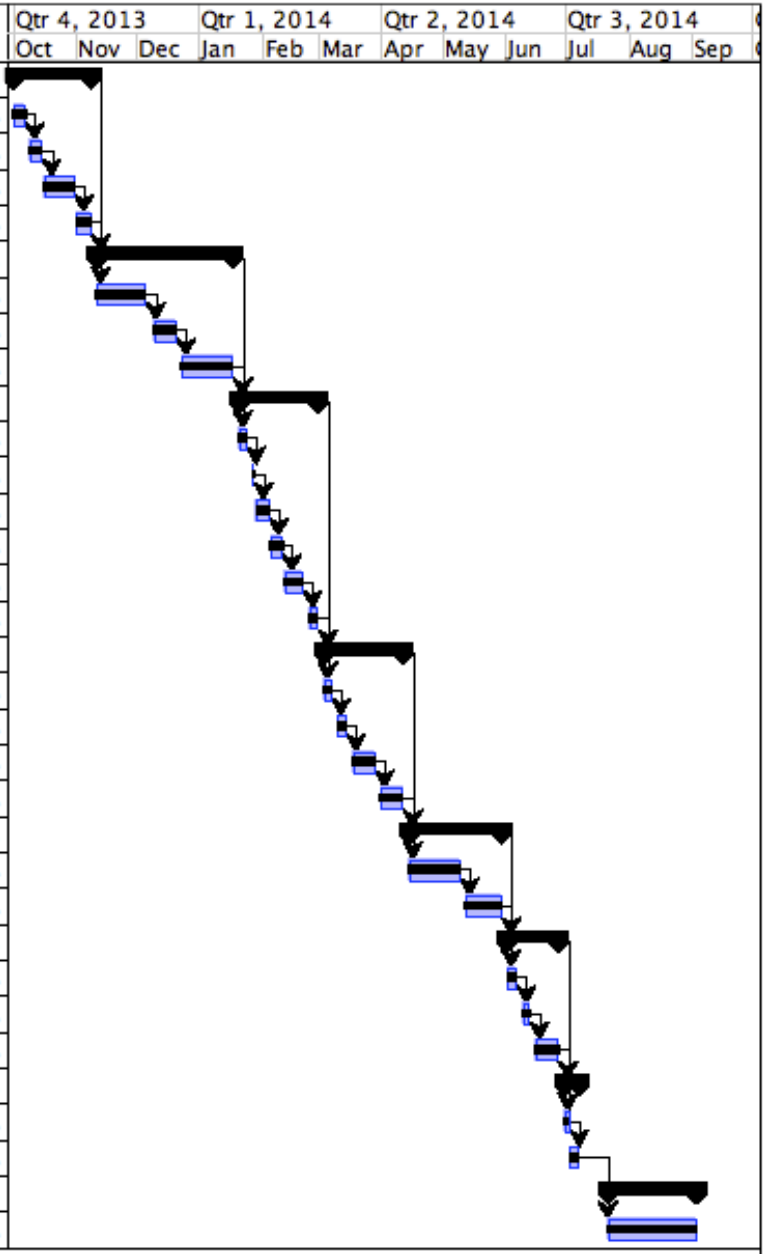


Figura A.2: Diagrama de Gantt con la planificación detallada del proyecto

A.3. Recursos

En esta sección se distinguen los diferentes recursos usados para la realización del Trabajo Fin de Grado:

- Recursos materiales:
  - 1 Servidor de pruebas Dell PowerEdge 2950, Procesador Intel Xeon, 20 GB RAM, Sistema Operativo: Debian
  - 2 PCs portátiles Macbook Pro 13-inch, Mid 2009, 8GB RAM; Sistema Operativo: Mac OS X
  - Bibliografía: Agile Web Development with Rails 4
  - Bibliografía: Everyday Rails Testing with RSpec: A practical approach to test-driven development
- Mano de obra directa:
  - 1 Graduado en Ingeniería Telemática: 11.55 ingenieros/mes
  - 1 Ingeniero Senior: 3.2 ingenieros/mes

A.4. Presupuesto de Proyecto

1. Autor: Alberto Martín Casado
2. Departamento: Ingeniería Telemática
3. Descripción del Proyecto:
   - Título: Desarrollo de una aplicación web para el proceso de contratación
   - Duración: 8 meses
   - Tasa de costes indirectos: 25 %.
4. Presupuesto total del Proyecto (valorado en Euros): euros. Ver tabla A.2
5. Subcontratación de tareas: no se especifican.
6. Otros costes directos del proyecto: no se especifican.

A.5. Marco regulador

En este proyecto se han utilizado herramientas de desarrollo de código libre como las mencionadas con anterioridad. La licencia de la que hace uso *Rails* (MIT) carece de copyright por lo que puede ser modificada. Lo mismo ocurre con la licencia GPL de *Ruby*, ampliamente usada en el mundo del software y que garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el software disponible.

Dado que la aplicación se ha desarrollado para uso privado de la organización, el código de la misma no será liberado. No obstante, de haberlo sido, el marco regulador a aplicar sería el que marcan las licencias de código libre utilizadas, siendo las mismas las que han de adoptarse.

Concepto	**Cantidad (€)**	**Coste €**	**% Proyecto**	**Dedicación (meses)**	**Depreciación (meses)**	**Total €**
Recursos materiales						
Servidor de pruebas	1	400	10	8	60	5.4
Ordenadores portátiles	2	1200	100	8	60	320
Bibliografía	2	35	100	8	60	9,34
Total						**334,74**
Mano de obra directa						
Graduado en Ing. Telemática	1 (11.55 ing/mes)	2.694,39	-	-	-	31.120,2
Ingenieros Senior	1 (3.2 ing/mes)	4.289,54	-	-	-	13.726,5
Total						**44.846,73**
Total de costes directos						**45.181,47€**
Costes indirectos						
Costes indirectos	-	-	-	-	-	25 %
Total						**11.295,37**
Total						**56.476,84 €**

Tabla A.2: Tabla de presupuesto

Apéndice B

Configuración del entorno de desarrollo

B.1. Introducción

En este apéndice se describe el proceso de instalación para la puesta a punto del entorno de desarrollo.

B.2. Características del equipo de desarrollo

La configuración hardware y de sistema operativo fue:

- CPU: 2,53 GHz Intel Core 2 Duo
- RAM: 8 GB 1067 MHz DDR3
- Sistema Operativo: OS X 10.9.4 (13E28)[28]

B.3. Configuración del framework Ruby on Rails

Se ha perseguido que el entorno de desarrollo cumpla lo siguiente:

- Configuración mínima y esencial
- Flexible con diferentes versiones de *Ruby*
- Fácil de instalar y configurar

Para su instalación, hemos seguido los siguientes pasos que se van a describir a continuación. Cabe destacar que se ha utilizado la última version de OS X, aunque la mayoría de versiones anteriores resultarían compatibles.

Los pasos son los siguientes:

- Instalar Homebrew[7] y Xcode Command Line Tools[29]
- Instalar *Ruby*
- Instalar *Rails*
- Configurar *Git*

B.3.1. Instalar Homebrew

En primer lugar, necesitamos instalar Homebrew. Homebrew nos permite instalar y compilar paquetes desde la fuente.

Homebrew funciona con un simple script de instalación. Cuando se solicita la instalación de Xcode CommandLine Tools, debemos instalarlo.

Abriremos el terminal y ejecutamos el siguiente comando:

```
ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

B.3.2. Instalar Ruby

Una vez tengamos Homebrew, vamos a utilizar rbenv[22] para instalar y gestionar las versiones de *en*Ruby.

Abriremos el terminal y ejecutaremos los siguientes comandos:

```
brew install rbenv ruby-build
echo 'if which rbenv > /dev/null; then eval "$(rbenv init -)"; fi'
>>~/.bash_ profile
source ~/.bash_ profile
rbenv install 2.0.0
rbenv global 2.0.0
ruby -v
```

Con estos pasos tendremos instalados rbenv y la versión que escojamos de *Ruby*. Por conveniencia, se ha escogido la versión 2.0.0 para todos los equipos y para evitar incompatibilidades.

B.3.3. Instalar Rails

Instalar *en*Rails es la parte más sencilla. Simplemente debemos ejecutar el siguiente comando:

```
gem install rails -v 4.0.1
```

Con esto, seleccionamos la versión de Rails. Hemos escogido la versión 4.0.1 de nuevo, para evitar incompatibilidades entre equipos, gemas y configuraciones.

B.3.4. Configurar Git

Para la instalación y configuración del repositorio, seguiremos los siguientes pasos:

- Instalar Git
- Añadir las llaves SSH al repositorio
- Descargamos la estructura
- Creamos la rama de desarrollo

B.3.4.1. Instalar Git

Dado que posteriormente hemos instalado Homebrew, será tan sencillo como ejecutar lo siguiente:

```
brew install git
```

B.3.4.2. Añadir las llaves SSH al repositorio

En primer lugar comprobamos si nuestro equipo dispone de llave SSH. Podemos comprobarlo ejecutando lo siguiente:

```
cat ~/.ssh/id_rsa.pub
```

Si aparece una larga cadena que comience por `ssh-rsa` o `ssh-dsa` simplemente necesitamos copiar la llave. De no ser así, generaremos una con el siguiente código:

```
ssh-keygen -t rsa -C "$youremail"
```

Gracias al uso de la interfaz web Gitlab[6], añadir la llave SSH que hemos obtenido o copiado es muy sencillo. Basta con navegar en nuestro perfil hasta el apartado SSH Keys y añadirla.

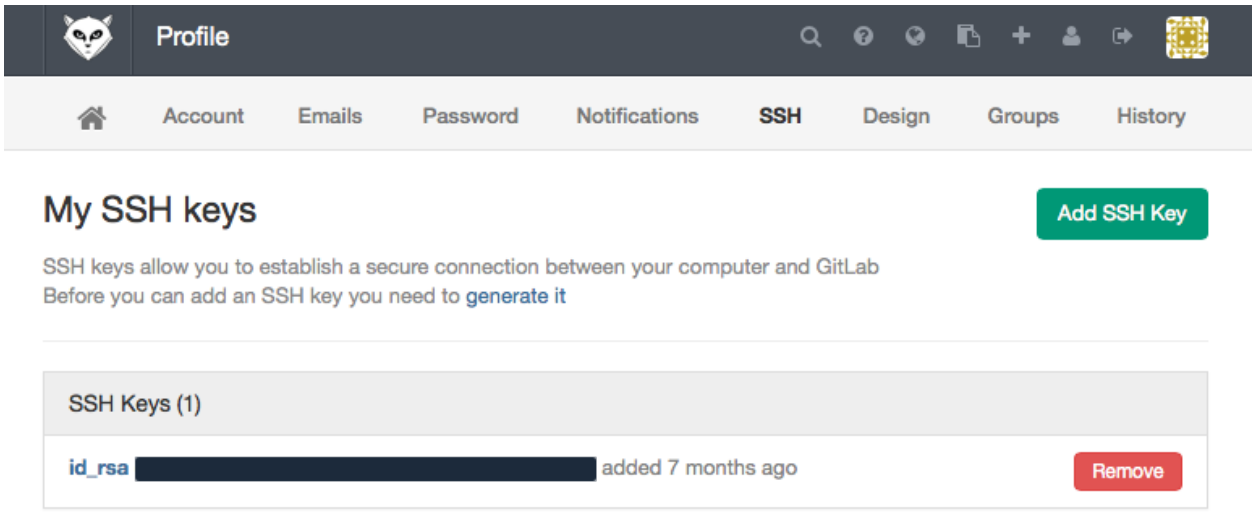


Figura B.1: Interfaz web Gitlab donde añadir la llave SSH

El servidor Git sobre el que trabajamos gestiona los usuarios gracias al sistema de autentificación LDAP (Lightweight Directory Access Protocol), el mismo que usa la empresa, por lo que la autentificación se realiza haciendo uso de ese módulo. El administrador del repositorio simplemente ha de añadir al usuario en cuestión al proyecto, y gracias a la llave SSH se puede establecer una conexión segura, asignando al usuario el rol de desarrollador.

B.3.4.3. Descargamos la estructura

Una vez tengamos acceso al servidor, descargamos la estructura y/o rama principal ejecutando el siguiente comando en el directorio que deseemos:

```
git clone git@gitlab.nombrededominio.org:path/nombre-del-repositorio.git
```

B.3.4.4. Creamos la rama de desarrollo

Para el desarrollo del código se han creado varias ramas o *en*branches. Esto permite mantener un control de versiones estables sin miedo a estropear nada. El procedimiento es sencillo:

```
git branch nueva-rama
git checkout nueva-rama
git push origin nueva-rama
```

Con esto habremos creado una nueva rama de desarrollo limpia y vacía. Posteriormente, se hará uso de otros comando de git como `clone` o `merge` para juntar los cambios de las diferentes ramas.

Apéndice C

Configuración de la gema Devise

C.1. Introducción

En este apéndice se describe la configuración que se ha seguido para dotar del servicio de autentificación a la aplicación.

C.2. ¿Qué es devise?

Devise es una gema de autentificación flexible para *Rails*. Se compone de 10 módulos que realizan lo siguiente:

- Encripta y almacena una contraseña en la base de datos para validar la autenticidad de un usuario mientras se autentifica. La autentificación se puede realizar tanto a través de peticiones POST o autentificación HTTP básica.
- Añade compatibilidad con OmniAuth (https://github.com/intridea/omniauth).
- Envía mensajes de correo electrónico con instrucciones para confirmar y verificar si una cuenta que se ha confirmado.
- Reestablece la contraseña del usuario y envía las instrucciones de restablecimiento.
- Maneja el registro de usuarios y permite editar y destruir cuentas.
- Gestiona tokens para recordar usuarios a través de cookies guardadas.
- Gestiona los registros a través de timestamps y direcciones IP.
- Comprueba la actividad de las sesiones y las cierra si no detecta actividad en un período de tiempo especificado.
- Proporciona validaciones de correo electrónico y contraseña.
- Bloquea una cuenta después de un número determinado de intentos de inicio de sesión fallidos. Se puede desbloquear a través de email o después de un período de tiempo especificado.

C.3. Instalación

Para la instalación, nos moveremos por el directorio de la aplicación y en el fichero Gemfile añadiremos la línea `gem 'devise'`

A continuación, en el terminal, ejecutaremos:

```
bundle install
rails g devise:install
```

Y habremos instalado la gema. A continuación configuraremos las diferentes opciones.

C.3.1. Modelo

Devise funciona sobre la arquitectura *MVC*, pero nos proporciona multitud de configuraciones posibles añadidas. Lo primero, generaremos el *modelo*. Para ello correremos la siguiente migración:

```
rails g DeviseCreateUsers
rake db:migrate
```

Con esto, generamos un archivo `db/migrate/YYYYMMDDxxx_devise_create_users.rb`. Esto nos permitirá añadir usuarios nuevos a nuestra aplicación. La configuración por tanto será la siguiente:

```
class DeviseCreateUsers < ActiveRecord::Migration
  def change
    create_table(:users) do |t|
      ## Database authenticatable
      t.string :email,              :null => false, :default => ""
      t.string :encrypted_password, :null => false, :default => ""
      ## Recoverable
      t.string   :reset_password_token
      t.datetime :reset_password_sent_at
      ## Rememberable
      t.datetime :remember_created_at
      ## Trackable
      t.integer  :sign_in_count, :default => 0, :null => false
      t.datetime :current_sign_in_at
      t.datetime :last_sign_in_at
      t.string   :current_sign_in_ip
      t.string   :last_sign_in_ip
      t.timestamps
    end
    add_index :users, :email,                :unique => true
    add_index :users, :reset_password_token, :unique => true
  end
end
```

C.3.2. Controlador

Una vez creado el *modelo*, definimos su funcionamiento en el *controlador* siguiente. La función `:authenticate_user!` comprobará en todo momento que el usuario se haya autentificado, evitando problemas de seguridad. Se declarará en todos los *controladores*. Para éste caso particular será:

```
class UsersController < ApplicationController
  before_filter :authenticate_user!

  def index
    authorize! :index, @user, :message => 'Not authorized as an administrator.'
    @users = User.all
  end

  def show
    params[:id] = current_user.id
    @user = User.find(params[:id])
  end

  def update
    authorize! :update, @user, :message => 'Not authorized as an administrator.'
    @user = User.find(params[:id])
    if @user.update_attributes(params[:user], :as => :admin)
      redirect_to users_path, :notice => "User updated."
    else
      redirect_to users_path, :alert => "Unable to update user."
    end
  end

  def destroy
    authorize! :destroy, @user, :message => 'Not authorized as an administrator.'
    user = User.find(params[:id])
    unless user == current_user
      user.destroy
      redirect_to users_path, :notice => "User deleted."
    else
      redirect_to users_path, :notice => "Can't delete yourself."
    end
  end
end
```

C.3.3. Vista de inicio de sesión

Finalmente, se generará la *vista* para la creación o inicio de sesión de usuarios.

```
<div class ="block-info-login">
        <p>To fill in the application form you must create an account,
        in order to modify your data in the future.</p>
        <p>If you applied in the past, please use your account instead
        of opening a new one.</p>

</div>
<div class="row-fluid">
<div class="span6" id="signup-form">

     <%= simple_form_for(resource, :as => resource_name,
     :url => registration_path(resource_name),
     :html => {:class => 'form-signin' }) do |f| %>

  <h2 class="form-signin-heading">Sign up</h2>

  <%= f.error_notification %>
    <%= f.input :email, :required => true, :autofocus => true %>
    <%= f.input :email_confirmation, :required => true %>
    <%= f.input :password, :required => true %>
    <%= f.input :password_confirmation, :required => true %>
    <%= f.button :submit, "Sign Up", :class => 'btn-primary' %>
  <% end %>

  </div>
  <div class="span6" id="signin-form">

    <%= simple_form_for(resource, :as => resource_name,
    :url => session_path(resource_name),
    :html => {:class => 'form-signin' }) do |f| %>

  <h2 class="form-signin-heading">Sign in</h2>

    <%= f.input :email, :autofocus => true %>
    <%= f.input :password %>
    <%= f.input :remember_me, :as => :boolean if devise_mapping.rememberable? %>
    <%= f.button :submit, "Sign in", :class => 'btn-primary' %>
    <hr />
    <%= link_to "Forgot your password?", new_password_path(resource_name) %>
  <% end %>

  </div>
</div>
```